INTELLIGENT CONTROL AND COOPERATION FOR MOBILE ROBOTS

by

PETRU EMANUEL STINGU

Presented to the Faculty of the Graduate School of

The University of Texas at Arlington in Partial Fulfillment

of the Requirements

for the Degree of

DOCTOR OF PHILOSOPHY

THE UNIVERSITY OF TEXAS AT ARLINGTON

December 2011

ACKNOWLEDGEMENTS

This work was supported by the following grants: NSF grant ECCS-1128050, ARO grant W91NF-05-1-0314, AFOSR grant FA9550-09-1-0278, and DARPA Phase I SBIR grant 08SB2-0934.

November 21, 2011

ABSTRACT

INTELLIGENT CONTROL AND COOPERATION

FOR MOBILE ROBOTS

PETRU EMANUEL STINGU

The University of Texas at Arlington, 2011

Supervising Professor: FRANK L. LEWIS

The topic discussed in this work addresses the current research being conducted at the Automation & Robotics Research Institute in the areas of UAV quadrotor control and heterogenous multi-vehicle cooperation. Autonomy can be successfully achieved by a robot under the following conditions: the robot has to be able to acquire knowledge about the environment and itself, and it also has to be able to reason under uncertainty. The control system must react quickly to immediate challenges, but also has to slowly adapt and improve based on accumulated knowledge.

The major contribution of this work is the transfer of the ADP algorithms from the purely theoretical environment to the complex real-world robotic platforms that work in real-time and in uncontrolled environments. Many solutions are adopted from

those present in nature because they have been proven to be close to optimal in very different settings.

For the control of a single platform, reinforcement learning algorithms are used to design suboptimal controllers for a class of complex systems that can be conceptually split in local loops with simpler dynamics and relatively weak coupling to the rest of the system. Optimality is enforced by having a global critic but the curse of dimensionality is avoided by using local actors and intelligent pre-processing of the information used for learning the optimal controllers. The system model is used for constructing the structure of the control system, but on top of that the adaptive neural networks that form the actors use the knowledge acquired during normal operation to get closer to optimal control. In real-world experiments, efficient learning is a strong requirement for success. This is accomplished by using an approximation of the system model to focus the learning for equivalent configurations of the state space. Due to the availability of only local data for training, neural networks with local activation functions are implemented.

For the control of a formation of robots subjected to dynamic communication constraints, game theory is used in addition to reinforcement learning. The nodes maintain an extra set of state variables about all the other nodes that they can communicate to. The more important are trust and predictability. They are a way to incorporate knowledge acquired in the past into the control decisions taken by each node. The trust variable provides a simple mechanism for the implementation of reinforcement learning. For robot formations, potential field based control algorithms

are used to generate the control commands. The formation structure changes due to the environment and due to the decisions of the nodes. It is a problem of building a graph and coalitions by having distributed decisions but still reaching an optimal behavior globally.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

CHAPTER 1

INTRODUCTION AND CONTRIBUTIONS

1.1. Introduction

During recent years, the contributions of unmanned systems to military and civilian operations continue to increase, reducing the risk to human life. Unmanned aircraft vehicles (UAV) and unmanned ground vehicles (UGV) can play an important role in missions such as reconnaissance and surveillance, precision target location, signals intelligence, and digital mapping. Because of the miniaturization of sensors and communication equipment, smaller aircraft can now perform the same mission that would have required a bigger aircraft and in some cases a human pilot on-board a few years ago. This has obvious advantages in terms of costs and logistics, but also it enables completely new missions, such as operation at street level in urban environments.

Today the level of autonomy is generally small. Human pilots have to provide high-bandwidth commands to the vehicle they control and cannot maintain strict control of an entire formation of such vehicles. This work presents algorithms that can improve the autonomy of real robotic platforms that perform in an uncontrolled environment, individually or in a formation by using reinforcement learning. The intelligent vehicles

and formations require only high-level, low-bandwidth commands. This way, the pilot is not required to have special training and an in-depth knowledge about the internal structure of the robots it controls.

Chapter 2 gives a succinct description of the mechanisms used to store knowledge in general and the control structures that can make use of this knowledge. All these mechanisms are inspired from nature. The robots have to react quickly to external stimuli but also have to adapt to changes in the environment and in their own parameters. At a higher level they have to plan and accomplish a mission in a distributed manner. These processes happen at different time scales and are separated into different controllers organized hierarchically.

Chapter 3 develops the system models of a quadrotor and ground robots. These models are used in the following chapters for the development of adaptive controllers and for guiding and focusing the reinforcement learning process.

Chapter 4 proposes a new parameterization for the attitude error of the quadrotor that decouples tilt from yaw and allows two different controllers to act independently for closing the two loops. The attitude error is expressed in body coordinates and creates the possibility for a linear mapping between the error and the actuators. A certain amount of tilt is parameterized the same for any yaw angle. This aspect is important for focusing learning. Even more, the pilot commands are relative to his coordinates frame and not to those of the vehicle.

Chapter 5 applies the principles of approximate dynamic programming (ADP) to the control of a quadrotor helicopter platform flying in an uncontrolled environment and

subjected to various disturbances and model uncertainties. ADP is based on reinforcement learning using an actor-critic structure. Due to the complexity of the quadrotor system, the learning process has to use as much information as possible about the system and the environment. Various methods to improve the learning speed and efficiency are presented. Neural networks with local activation functions are used as function approximators because the state-space cannot be explored efficiently due to its size and the limited time available. The complex dynamics is controlled by a single critic and by multiple actors thus avoiding the curse of dimensionality. After a number of iterations, the overall actor-critic structure stores information (knowledge) about the system dynamics and the optimal controller that can accomplish the explicit or implicit goal specified in the cost function.

Chapter 6 describes an implementation of a distributed control algorithm using potential field methods to control a fleet of small nonholonomic ground vehicles. Each robot has limited communication resources and needs to play a network formation game in order to build a communication graph. This type of game is a hybrid between coalitional graph games and non-cooperative games where the player's strategies are to select one or more links to form or break. A trust variable was introduced that eliminates the need of negotiation between nodes that would require extra communication resources and successive iterations. Trust is also used to control the behavior of the robots and to help them avoid misbehaving agents by using reinforcement learning.

1.2. Contributions

This work makes several contributions.

In the field of computational intelligence, it develops approximate dynamic programming algorithms for complex dynamical systems with a large number of discrete-time, continuous valued states by using multiple local actors and a single global critic. In the past, the high dimensionality of the state space for such systems made the problem intractable.

In the field of control engineering, it develops on-line, real-time, sub-optimal adaptive control algorithms. Generally, optimal control algorithms were developed off-line, and adaptive control algorithms were not optimal.

In the field of machine learning, it develops various methods that focus learning for similar contexts to the same region in the neural network in order to improve the learning speed. This is accomplished by using knowledge about the system model in order to do some preprocessing of the neural network inputs. For the particular case of the quadrotor, it introduces a new parameterization of the attitude error in the body frame instead of using Euler angles. This parameterization provides a better decoupling of the control loops and focuses learning.

In the field of game theory it introduces the concept of trust and its implementation using reinforcement learning.

CHAPTER 2

NATURE–INSPIRED CONTROL OF AUTONOMOUS ROBOTS

Autonomous robots are robots which can perform desired tasks in unstructured environments without requiring continuous human guidance. Most of the times, the dynamics of the robot itself can be described analytically. Unfortunately, in many robotic applications, it is difficult if not impossible to obtain a precise mathematical model of the environment and its interaction with the robot through actuators and sensors. The lack of complete and precise knowledge about the environment limits the applicability of conventional control system design to the domain of autonomous robotics. Some of the requirements for a robot to successfully achieve autonomy are the possibility to acquire knowledge about the environment and itself, to reason under uncertainty and to have learning capabilities in order to adapt to the environment based on accumulated experience.

2.1. Neuro-Fuzzy Control

Efficient control algorithms for autonomous robots should imitate the way humans are operating manned or similar vehicles. When making decisions, humans tend to work with vague or imprecise concepts that can often be expressed linguistically. Lotfi Zadeh has proposed one way to model this decision making process by

introducing the fuzzy set theory in the field of control [1]. Fuzzy logic is particularly suited for problems in which the data, the objectives and the constraints are too complex or too ill-defined to admit a precise mathematical analysis.

Neural networks were developed as an attempt to realize mathematical models of brain-like systems. The key advantage is their ability to learn from examples instead of requiring an algorithmic development from the designer. They can generalize to new situations. Once a neural network has been trained for a set of data, it can interpolate and produce answers for the cases not present in the training set.

While fuzzy logic can be used to represent knowledge in a human-readable form and to use it for reasoning, neural networks allow adaptation and learning in dynamic environments under varying conditions. Neuro-fuzzy techniques combine the advantages of both methods by having neural networks adapt the knowledge base of the fuzzy logic systems or fuzzy systems tune the weights of neural networks. These relatively simple control methods can be used to successfully implement complex intelligent autonomous robots, robust to uncertainties in their own model, in the environment and in the readings from the sensors.

Despite the differences between neural networks and fuzzy logic systems, they can actually be unified at the level of the universal function approximator. Both of them define a nonlinear function $y = f(x)$ from inputs to outputs and if designed properly satisfy the "universal approximation property" [2], [3]. Then, for any continuous function $\psi(x)$ defined on a closed and bounded set and an arbitrary number $\varepsilon > 0$, there exists a neural network or a fuzzy system $f(x)$ such that

$$\sup_x \left| f(x) - \psi(x) \right| < \varepsilon \ . \tag{1}$$

This property does not say how to build the neural net or the fuzzy system. It simply shows that with the proper structure and with enough tuning it is possible to approximate a continuous function with an error that can be made as small as desired.

The multilayer perceptron should be viewed as a nonlinear network whose nonlinearity can be tuned by changing the weights, biases and the parameters of the activation functions. The fuzzy logic system is also a tunable nonlinearity whose shape can be tuned by changing the membership functions. In both cases, it is possible to use gradient methods for tuning. The back-propagation training method [4] is well-known from the neural networks field and inspired gradient training of fuzzy systems.

Some radial basis function neural networks are functionally equivalent to some standard fuzzy systems in the sense that given the same inputs, they will produce the same outputs. This can be shown on a normalized radial basis functions (RBF) neural net that has the same number of neurons on the hidden layer as the number of rules in the fuzzy system with product inference and centroid defuzzification. A simple example for a two-input fuzzy system is shown in Figure 1. A RBF neural net structure that can process the information in a similar way is shown in Figure 2.

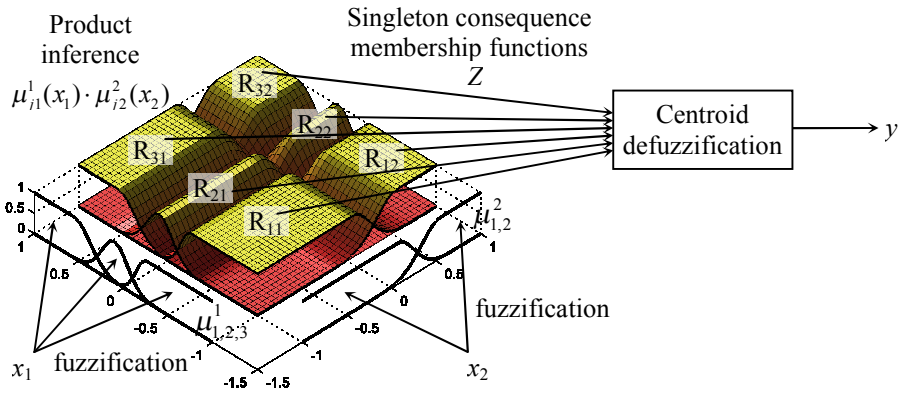


Figure 1. Example of a two-variable fuzzy logic system

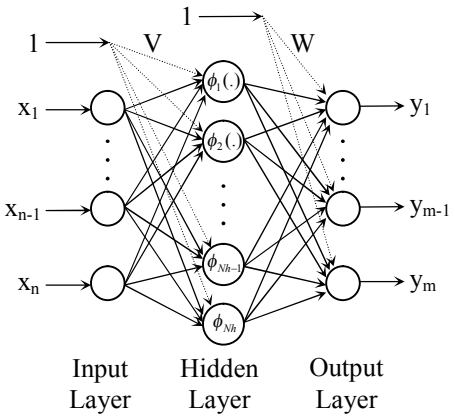


Figure 2. Multi-layer neural network

The output of a fuzzy system with n input variables, possible different number of membership functions for each input variable, product inference, control representative values $z_{i,j1,j2,\ldots,jn}$ and centroid defuzzification is the following:

$$y = \frac{\sum_{j1,j2,\ldots,jn} z_{i,j1,j2,\ldots,jn} \prod_{k=1}^{n} \mu_{jk}^{k}(x_k)}{\sum_{j1,j2,\ldots,jn} \prod_{k=1}^{n} \mu_{jk}^{k}} \tag{2}$$

The k^{th} output of an unnormalized RBF neural network with N_h neurons on the hidden layer is

$$y_k = \sum_{i=1}^{n} w_{ik} \cdot \varphi_i(\mathbf{x}) \tag{3}$$

while the output of a multi-layer neural network is

$$\mathbf{y} = W^T \boldsymbol{\varphi}(V^T \mathbf{x}). \tag{4}$$

Because of their properties of learning, recall, function approximation, generalization, classification, association, pattern recognition and clustering, neural networks and fuzzy logic systems can be used to solve a large set of problems. They are successfully used in all types of autonomous robots.

2.2. Control Hierarchy

We have seen that autonomous robots can borrow traits from humans. Fuzzy logic copies the way they represent knowledge and the mechanisms of logical reason. Neural networks implement a simple model for the low-level organization and the physiological mechanisms relating to information processing in the nervous system. It is a well-known fact that nature has found the optimal solutions for many problems, over thousands and thousands of years of refinements. Humans have become the most autonomous of all living beings. Such a successful model should also suggest the control hierarchy to be used on autonomous robots. The organization of the human nervous system involved in movement can provide invaluable information on how to design an efficient hierarchical control architecture.

Consideration of the vast and complex system of structures and pathways involved in movement will begin at its lowest end, the spinal cord, where more basic motor control is localized. The most important human interoceptive reflex is the myotatic reflex [5], which originates from the neuro-muscular fibers. Its principal function is maintaining the joint position fixed and compensating external noise. In Figure 3, suppose that a load is applied to the joint. This will flex the joint, causing the stretching of the extensor muscle and also stretching of the spindle. This will increase the output of the neural stretch receptor neuron, which increases the output of the local motor neuron (LMN). The resulting increase in the contraction force will compensate the load. This local feedback allows the higher system to ignore the fluctuation in contraction required to maintain a certain joint extension. On a robot, the similar function is done by the low-level controllers for the motors. For example, they receive a reference speed and they have to keep the output speed equal with the reference, independent of the torque disturbances. Neural nets and fuzzy logic speed regulators have been successfully applied for motor control, yielding better results than normally used PID controllers.

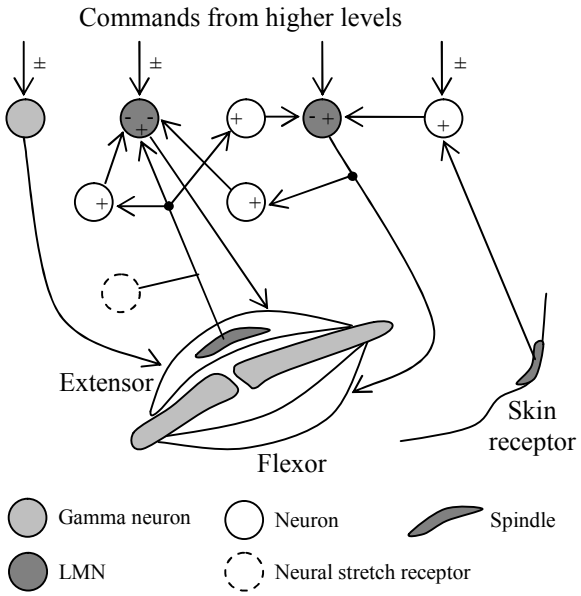


Figure 3. Joint with the principal neurons involved in reflex control

The conventional hierarchical control architecture that has been extensively used for mobile robots is shown in Figure 4. The robot builds the model of itself using two types of sensors. The proprioceptive sensors, like shaft encoders, give information about the internal state of the robot. The exteroceptive sensors, like a sonar, provide information about the state of the environment. Using this information, a planning algorithm generates a plan that will perform the given task in the given environment. The instructions from the plan are blindly executed by the lower-level motor control layer. This layer uses proprioceptors for local feedback, but does not monitor the environment. If an obstacle is suddenly detected by the exteroceptors, it takes a long time for the robot to react. This happens because the stimulus has to pass through the higher layers first. Modeling and planning usually involve a lot of computation and take a long time. The response of the overall system to a new configuration of the

environment is very slow. That is why robots implemented using this control hierarchy have to move with a very low speed in order to be able to avoid obstacles.

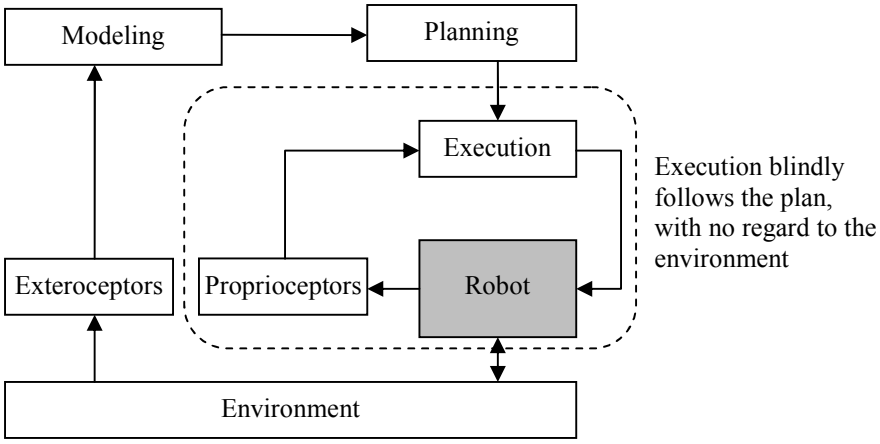


Figure 4. The low-level in the hierarchical architecture

A solution to the above problem can be searched in the way humans react to some external stimuli, for example to pain. A simple spinal reflex can be traced in Figure 3. When the pain receptor in the skin is excited, it fires a neuron in the LMN system, which in turn fires the LMN driving the flexor muscle. This operation removes the respective part of the body from danger, in a very fast and straightforward manner. There is no reaction expected from the higher layers of the nervous system before the motor neurons are fired. Still, the information from the skin receptor reaches the higher layers and a more intelligent measure is taken after a certain delay. The overall system now has a fast response to the environment. Reasoning about the necessary action can still be done and the movement can be corrected by the central nervous system through the direct inputs to the motor neurons. On mobile robots, the same type of reaction can be achieved by using low-level behaviors implemented in the execution layer. A simple

but very important modification [6] has to be done to the structure in Figure 4. The execution layer will not follow the planner commands blindly anymore, but will also receive information about the environment from the exteroceptive sensors, as shown in Figure 5. The behaviors (that can be easily implemented using fuzzy logic) will have a fast reaction to the environment, because it is included now in the low-level feedback loop. The planner will only have to select the right behavior or to combine behaviors corresponding to the current goal and the current environment. The rapidity of decisions is not critical anymore for the higher levels.

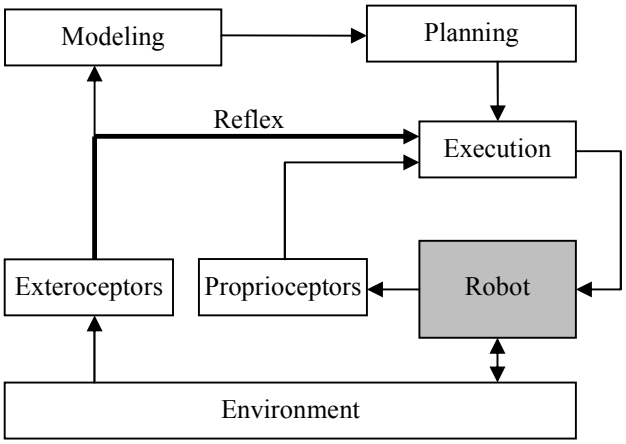


Figure 5. Implementation of behaviors based on the sensed environment

From the overall organization of the human motor nervous system [7], shown in Figure 6, and from the previous examples, a general principle can be extracted. First of all, (complete) sensorial data arrives to all the layers of control, starting from the lowest (the spinal cord) and up to the highest (the cerebral cortex). Each layer uses what information it needs and has certain autonomy in taking decisions, independent of the higher layer in the hierarchy. The latter will eventually correct or adjust the action

initiated by the lower layers. The reaction speed is the fastest for the simple layers close to the effectors (of the order of hundreds of milliseconds) and decreases as one rises through the hierarchy and where structures become more complex.

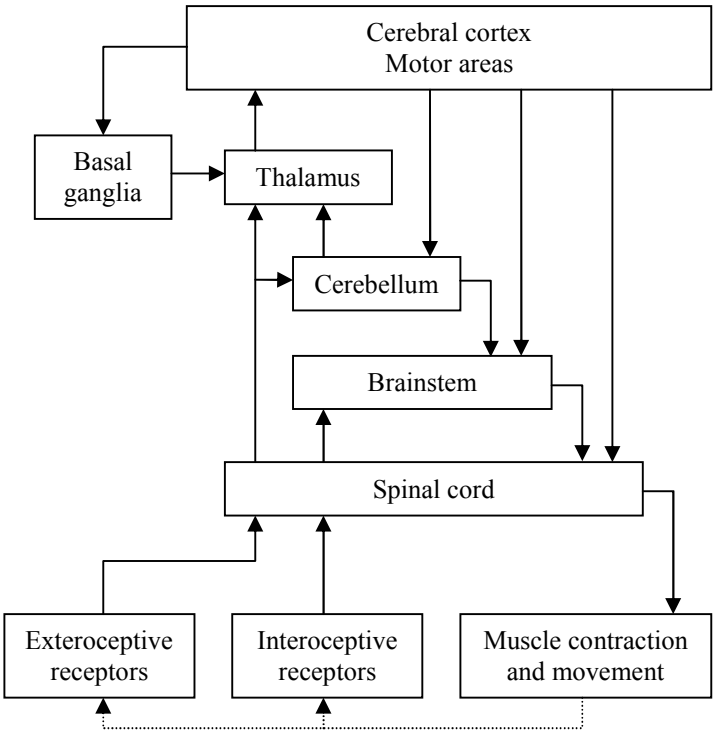


Figure 6. Summary of motor control in the human nervous system

In [8], Rodney A. Brooks has proposed a similar layered control architecture for mobile robots, organized into levels of competence. It is called the *subsumption architecture* and is based on incorporating lower levels of functionality into more general levels (Figure 7). Each level of competence includes as a subset each earlier level of competence. Practically the higher layers impose additional restrictions on the behaviors implemented in the lower layers. The lowest-level layer is layer 0. It is implemented and tested by its own. Layer 1 is then built on top of layer 0. It can read

information from layer 0 and can inject information into the internal interfaces of layer 0, suppressing the normal data flow. Level 1 of competence is achieved using layer 1 and the help of layer 0. The latter continues to run unaware of the fact that layer 1 interferes with its data paths when it wants to take control. Many layers of competence can be added one on top of the other using this mechanism. The functionality of the different layers for a mobile robot spawns across three main levels, as stated by Stephanou in [9] and by Lefebvre and Saridis in [10]: organization, coordination and execution.

The lower levels in the hierarchy have very fast reactions to sensor readings. They don't usually do complex processing on this information. The higher levels usually do sensor fusion and apply special algorithms to extract more abstract information about the environment. The decisions are taken with longer delays and with a much lower rate. It can be observed that there is a transition between almost continuous-time control at the lower levels, with high sample rates, and relatively slow discrete events at the highest levels. This transition can be exemplified with the difference between a fast PI controller for the wheels motors, and a discrete event controller for the implementation of the main plan of action.

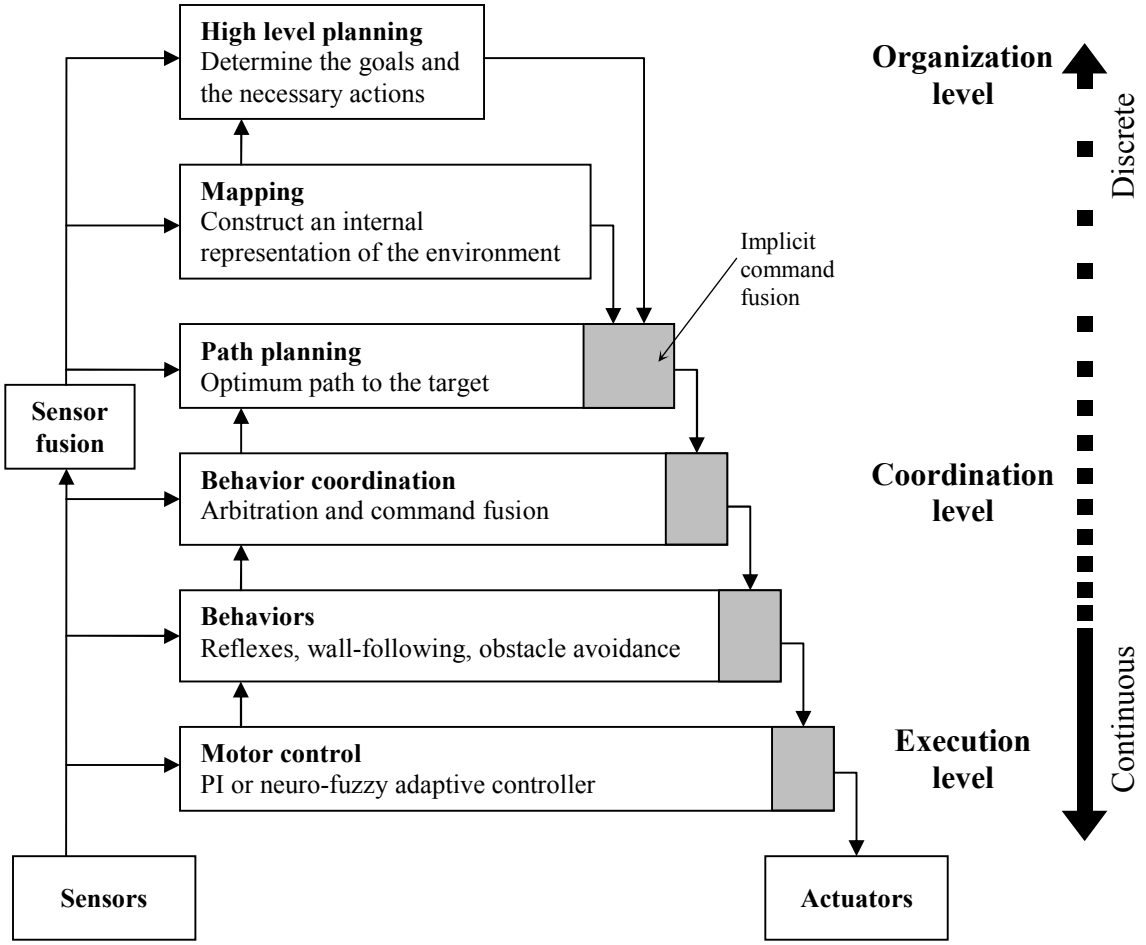


Figure 7. Subsumption architecture for robot control

2.3. Navigation Using Artificial Potential Fields

Any approach to control a dynamic system needs to use a model of the system to be controlled. In the case of a mobile robot, the system consists of the robot itself plus the environment in which it operates. It is easy to obtain the model of the robot on its own, but unfortunately the situation is different if we consider the robot to be embedded in a real-world, unstructured environment, that is characterized by uncertainty which is

difficult to model or to quantify. The mapping problem is generally regarded as one of the most important problems in the pursuit of building truly autonomous mobile robots.

Map-based navigation calls upon three processes:

- Map learning - the process of memorizing the data acquired by the robot during exploration in a suitable representation.
- Localization - the process of deriving the current position of the robot within the map.
- Path planning - the process of choosing a sequence of actions in order to reach a goal, starting from the current position.

Map learning and localization are interdependent. Building a map requires the position of the robot to be estimated relative to the incomplete map while localization requires that the map exists. Path planning is rather independent of the other two and takes place once the map and the robot position are already available.

Various map representations have been used in the robotics literature. They are adequate or not depending on the task and the characteristics of the robot and the environment.

Artificial potential field methods for obstacle avoidance have gained increased popularity among researchers in the field of mobile robots. The idea of imaginary forces acting on a robot has been suggested by Andrews and Hogan [11] and Khatib [12]. The approach used to generate the artificial potential fields is to have obstacles exert repulsive forces onto the mobile robot, while the target applies an attractive force to it. The resultant force $\vec{F}$ determines the direction and speed of travel for the robot. The

method is simple and elegant, yielding acceptable results with simple and quick implementations and without requiring many refinements. The moving robot can come in a variety of shapes and sizes. To simplify the development of the solution, the robot can be represented as a point while the obstacles and physical workspace boundaries are transformed by increasing their size with a value related to the dimension of the robot.

The potential is a scalar field whose negative gradient is a vector field of conservative forces. Any point of the force field will represent the resultant force obtained by summing the effects of the attraction force of the target and the repulsion forces of the obstacles:

$$\mathbf{F}(\mathbf{r}) = \mathbf{F}_{target}(\mathbf{r}) + \sum_{i=1}^{n} \mathbf{F}_{obs_i}(\mathbf{r}) = -\nabla V(\mathbf{r}) \tag{5}$$

For mobile robots applications, the potential fields are usually used in a 2-dimensional space:

$$\begin{aligned} F_x(x,y) &= -\frac{\partial V(x,y)}{\partial x} \\ F_y(x,y) &= -\frac{\partial V(x,y)}{\partial y} \end{aligned} \tag{6}$$

The trajectory is generated continuously for any position (x, y) of the robot following the direction of the steepest descent. It can be considered an optimization problem that searches for the point of minimum potential.

Let's consider the model of a car-like vehicle with no mass. It is a nonholonomic system, so it will most likely deviate from the ideal trajectory resulted from the potential field. However, the potential field implicitly provides a feedback

mechanism. The **F** force is generated for any position of the robot in such a way that it will return close to the ideal trajectory.

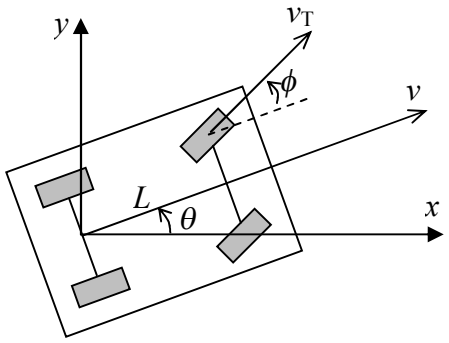


Figure 8. Model of a car-like robot

The robot (Figure 8) has the following dynamics:

$$\begin{aligned} \dot{x} &= v_T \cos\varphi \cos\theta \\ \dot{y} &= v_T \cos\varphi \sin\theta \\ \dot{\theta} &= \frac{v_T}{L}\sin\varphi \end{aligned} \tag{7}$$

with (x, y) the position, θ the heading angle, v_T the wheel speed, L the wheel base, and ϕ the steering angle.

A potential field corresponding to the following environment is generated:

- Goal: a constant force will attract the robot to the target.

$$(x_G, y_G) = (10,10)$$

$$r = \sqrt{(x_G - x)^2 + (y_G - y)^2} \tag{8}$$

$$F_{Gx}(x, y) = K_G \frac{x_G - x}{r}, F_{Gy}(x, y) = K_G \frac{y_G - y}{r} \tag{9}$$

- Circular obstacles: a force similar to the electrostatic force will repel the robot from the obstacles.

$(x_1, y_1) = (3,3)$ for the first obstacle

$(x_2, y_2) = (7,2)$ for the second obstacle

$$F_{ix}(x,y) = -K_i \frac{x_i - x}{\left(\max\left\{(r-a_i), b_i\right\}\right)^2}, \quad F_{iy}(x,y) = -K_i \frac{y_i - y}{\left(\max\left\{(r-a_i), b_i\right\}\right)^2}, \quad (10)$$

where r is the distance between the robot and the goal or the obstacle, a_i is the radius of the obstacle and b_i limits the relative height of the obstacle. The total force that acts on the robot is

$$\begin{aligned} F_x &= F_{Gx} + F_{1x} + F_{2x} \\ F_y &= F_{Gy} + F_{1y} + F_{2y} \end{aligned} \quad (11)$$

and the angle of the force as seen from the robot is

$$\alpha = \tan^{-1}\left(\frac{F_y}{F_x}\right) \quad (12)$$

with corrections for the correct quadrant.

The initial position for the robot is $(x_0, y_0) = (0,0)$ and the initial heading angle is $\varphi_0 = \frac{\pi}{6}$. The speed v_T is often kept constant. The steering angle may be generated by the controller based on the difference between the heading angle θ and the angle α of the resultant force acting on the robot (Figure 9). In this example, a proportional controller is implemented:

$$\varphi = K(\theta - \alpha) \quad (13)$$

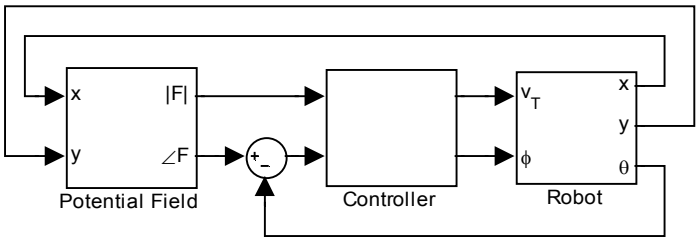


Figure 9. Closed-loop control based on the potential field

It can be seen from the potential field representation (Figure 10) that the robot will try to follow the direction of the steepest descent, which is the direction of the resultant force. There are a few issues that can prevent the robot to reach the target:

- trap situations due to local minima
- no passage between closely spaced obstacles
- oscillations in the presence of obstacles or narrow passages

Some of the problems can be solved by modifying the height or dimension of the obstacles, by changing the type of force used to represent them or by using an intelligent controller in the structure from Figure 9.

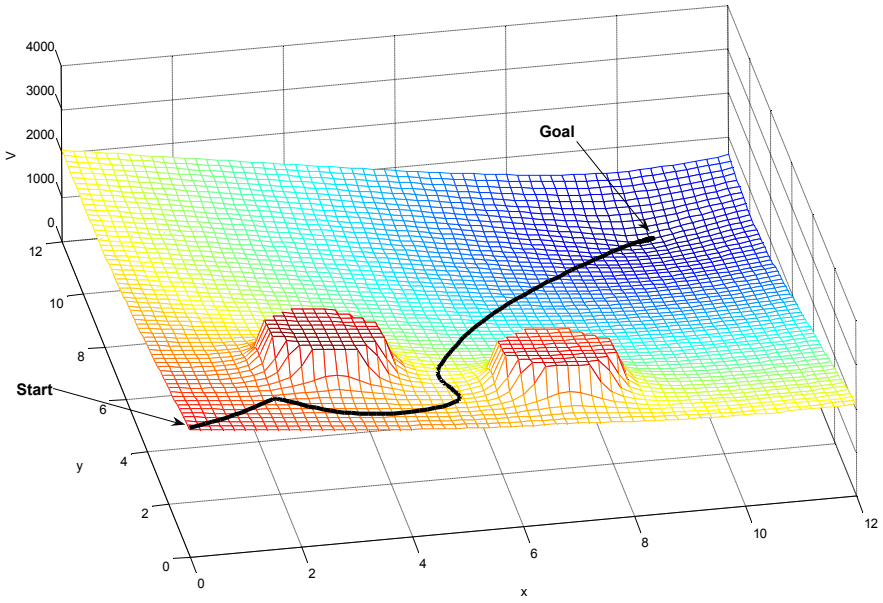


Figure 10. The trajectory of the robot using the potential field

2.4. Combination and Coordination of Behaviors

The operational characteristics of unmanned vehicles include the following:

- Perception: acquire knowledge about the environment using sensors and extracting meaningful information to be used in later tasks
- Intelligence: operate for a considerable amount of time without human intervention while accomplishing useful tasks
- Action: in general, move between certain points.

One of the most common applications of fuzzy logic for autonomous robotics is to implement individual behaviors. Considering the environment uncertainty that is difficult if not impossible to model, effective control algorithms for autonomous

navigation should imitate the way humans are operating various vehicles. Fuzzy logic allows a suitable knowledge representation of inherently vague notions achieved through IF-THEN rules. These rules contain linguistic information that describes the problem in a simple and fast manner. In many applications of fuzzy logic, a mathematical model of the dynamics of the vehicle is not needed. The only requirement for the design of the inference engine is the heuristic control knowledge related to the specific problem, usually obtained from a human who knows how to handle the system. Tuning is done by trial and error methods. Because of their interpolative nature, fuzzy controllers generate smooth movement for the robots and provide robustness and a graceful degradation of performance when confronted with noise and other type of errors in the data obtained from the sensors.

Typically, fuzzy controllers consider a single goal. Isolated reactive behaviors are incapable of performing autonomous navigation in complex environments. However, more complex tasks can be accomplished through combination and cooperation of primitive behaviors. For the situations where there are two or more goals active simultaneously, there are two solutions:

- Build complex rules whose antecedents consider both goals at the same time.
- Write two sets of simple rules, each set specific to a single goal, and combine their output using some form of behavior arbitration or command fusion.

Arbitration is the process that leads to the activation of a specific behavior or that generates weights for multiple simultaneous behaviors. It decides which behaviors should influence the operation of the robot and how much.

The subsumption architecture implemented by Brooks in [8] is representative for fixed behavior arbitration. Because the robot was doing a single task, he used a fixed arbitration policy, built as a network of suppression and inhibition links. However, autonomous robots need to adapt to the environment and to perform multiple tasks. As a result, the arbitration strategies have to be dynamic, taking into consideration both the environment configuration and the current plan from the higher decisional levels. By using fuzzy logic to implement either fixed or dynamic arbitration, the transition between behaviors will be smooth and it will be possible to allow partial and concurrent activation of behaviors.

The most simple and obvious method used to fuse the commands coming from multiple behaviors consists in a switching scheme (Figure 11). The behaviors are activated only one at a time. The selected dominant behavior solely controls the robot until the next decision cycle, whereas the motor commands of the suppressed behaviors are completely ignored. The arbitration strategy determines which behavior is activated. This switching approach achieves a poor performance in the presence of multiple goals. A good example is the potential field navigation using a reactive behavior for the avoidance of obstacles that are not represented on the map. There are two goals: the global behavior tries to reach the target, while the reactive behavior has to avoid the obstacles. When the robot approaches an obstacle, the control is taken away from the

target-reaching behavior and given to the obstacle avoidance behavior. The latter can take the arbitrary decision of passing the obstacle through the left or through the right. Even if the goal of avoiding the obstacle is reached, the global solution to the navigation problem can suffer significantly if the wrong direction is chosen.

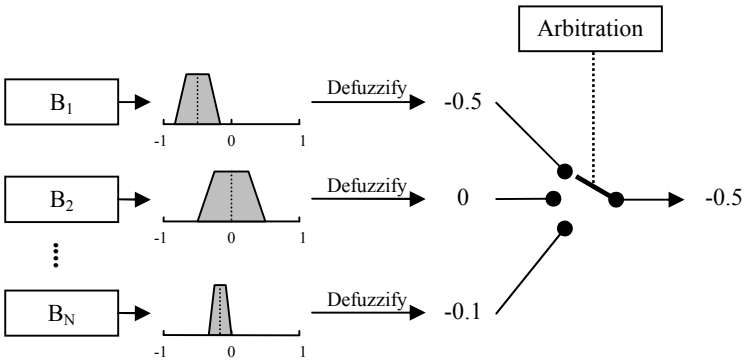


Figure 11. Switching scheme for command fusion

Parallel execution of multiple behaviors can overcome some of the limitations of the switching scheme. The final commands given to the robot can be calculated in two ways: by combining individual decisions (Figure 12) or by combining individual preferences (Figure 13).

The vector summation scheme is a good example of combining individual decisions. Considering the previous problem relating to the potential field navigation, we can identify two behaviors. Each one provides a vector characterizing the desired velocity and direction of movement. By summing the two (weighted) vectors, it is possible to obtain an intermediate speed and direction in order to reach both goals. This approach does not necessarily need fuzzy logic, although it can be easily implemented using weighted singletons as fuzzy outputs and center-of-gravity defuzzification.

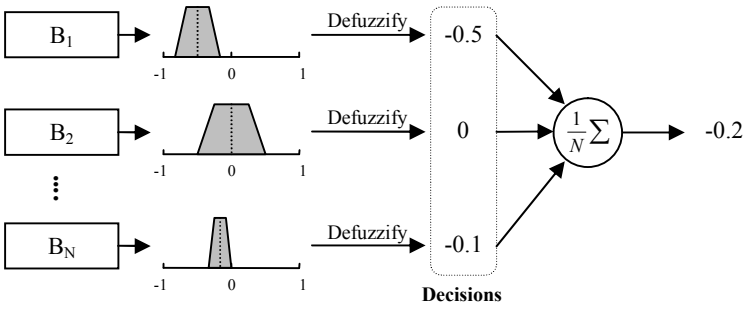


Figure 12. Command fusion by combining individual decisions

The real advantage of using fuzzy logic for command fusion becomes evident when combining individual preferences. A preference can be represented as a probability density function or as a fuzzy set [13]. It provides more information than a decision, which is a single (crisp) value. It gives information about an entire range of possible values and about how desirable they are for accomplishing the required task. Fuzzy logic has many different operators to perform combination and many defuzzification functions to perform decision. If the behaviors are also implemented using fuzzy logic, it is easy to have them output a fuzzy set instead of a crisp value, by eliminating the defuzzification phase. Even behaviors implemented using a different mechanism, but that output a PDF, can be easily accommodated.

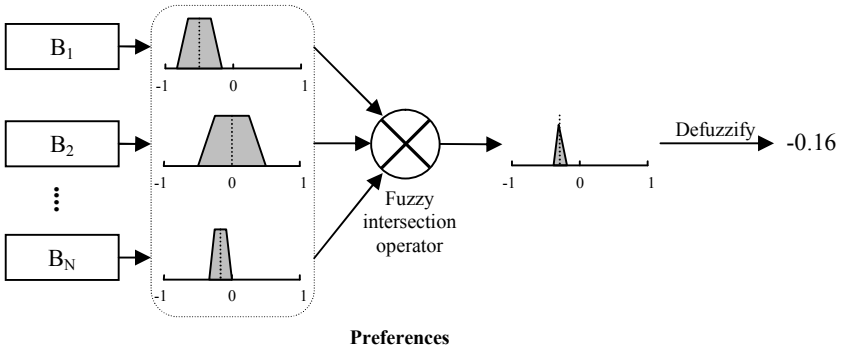


Figure 13. Command fusion by combining individual preferences

All the previous methods of command fusion suffer in some way or another when they have to handle competing behaviors that issue conflicting control commands. In this case, the resulting motor command from the compromise decision might be sub-optimal or even worse than any of the individual commands. An example could be the situation where the defuzzification results in the selection of a value that lies between two peaks of the combined fuzzy set. For a robot that has to avoid an obstacle by going to the left or to the right, this would make it go forward, straight into it. There is a need for extensions to basic fuzzy command fusion schemes capable of resolving conflicts among contradicting actions. A simple example is given in [14], where Yen proposes a different defuzzification approach, by replacing the center of gravity with centroid of largest defuzzification. This only considers the output fuzzy set with the largest area and completely ignores all the others, making the method similar to the majority voting schemes.

Context-dependent blending of behaviors is the most general type of behavior combination that can be realized using fuzzy logic. The method was suggested by Ruspini in [15] for the Flakey robot and later by Saffiotti in [16]. They reintroduce some form of behavior arbitration into fusion by having a set of higher-level supervisory fuzzy rules to activate and deactivate the individual fuzzy behaviors. As a consequence, the hierarchical behavior architecture is composed of two distinct layers. On the higher level, behavior coordination is achieved by means of supervisory fuzzy rules of the form:

IF *context* THEN *behavior*

As seen in Figure 14 (Saffiotti, [17]), each behavior generates preferences in order to reach its goal. There is a context of activation for each behavior. It describes the applicability and desirability of that particular behavior and also reflects the needs of higher-level goals. The preferences of all behaviors, weighted by the truth value of their contexts, are combined to obtain the collective preference. The crisp value for the associated command is obtained after defuzzification.

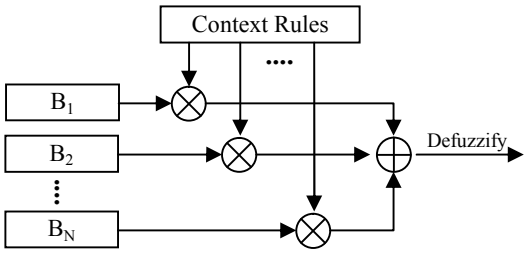


Figure 14. Context-dependent blending of behaviors

The decisions can be event-driven or goal-driven. For example, in the potential field navigation problem with a reactive behavior for unknown obstacle avoidance, the following event-driven rules can apply:

IF *obstacle-close* THEN *avoid-obstacle*

IF *not(obstacle-close) THEN go-to-target*

Depending on the value of the *obstacle-close* fuzzy variable, the behaviors are activated with various strengths: for extreme values, either *avoid-obstacle* or *go-to-target* is fully enabled; for intermediate values, both behaviors are partially enabled. The goal-driven approach can be used, for example, to sequence behaviors. If the robot has to pass through a set of waypoints, these rules can be used:

IF W_1 *not reached* THEN *go-to-*W_1

IF W_1 *reached* AND W_2 *not reached THEN go-to-*W_2

...

IF W_{N-1} *reached* AND W_N *not reached THEN go-to-*W_N

The path followed by the robot will be smooth due to fuzzy interpolation of behaviors. The direction of movement and the speed will not change suddenly when a waypoint is bypassed. The event-driven and the goal-driven blending of behaviors can be combined into an arbitrarily complex set of rules in order to represent a full plan of action.

Some standard algorithms used for mobile robot navigation can be improved or replaced by using neuro-fuzzy methods. While many implementations require the use of very precise sensors, full models of the systems and complicated mathematics, it can be observed in nature that the same actions can be realized by animals or humans using less precise sensorial information and without having access to or needing a mathematical model of the system or the environment. Fuzzy logic and neural networks can be used to implement a similar behavior for machines. Their full potential in this area has not been reached yet.

CHAPTER 3

MODELING OF AIR AND GROUND VEHICLES

This chapter presents the hardware testbeds used to validate the algorithms form Chapters 4, 5 and 6. It also develops the models of the quadrotor and the ground vehicles used in Chapters 5 and 6.

3.1. Hardware Implementation of the Quadrotor Platform

The testbed for the experiments consists in a quadrotor platform with on-board sensors and brushless motors, a ground computer with a radio basestation and a remote control (Figures 15–18). The mechanical platform is custom-built because the commercial versions available are either unreliable or too expensive. The outside perimeter is surrounded by carbon fiber rods to prevent damage to the rotors in case of collisions.

The entire electronic system (e.g. circuit boards) and the software are developed and built by the author in order to allow maximum flexibility when needed. The control algorithms are implemented in Simulink on the ground computer and run directly in the Matlab environment in normal or accelerated mode, allowing an instant transition between design and experiments. A special S-function block was created to allow the Simulink model to receive sensor data from the quadrotor and to send back the motor

commands. The block communicates via USB with the base-station module connected to the computer. The sample rate is limited at 50 Hz by the speed of the wireless data link. All radio communication is done in short packets with minimum latency in such a way that during the sample period of 20 ms the sensors are sampled on the quadrotor, the data is sent over radio to the base-station and then through USB to the Simulink model, the results of the control algorithm are sent back to the base-station and from there through radio to the quadrotor to control the servomotors.

The remote control uses the same two-way communication link to send commands to either the quadrotor or the computer (i.e. for reference inputs) and can also display important process variables to the pilot on the ground. Brushless DC motors with off the shelf speed controllers are used to improve the reliability and the repeatability of experiments. A remote-controlled safety switch was implemented in order to cut the power to the motors when the pilot is in the proximity of the unit and to prevent accidents due to faulty electronics, software or bad control algorithms.

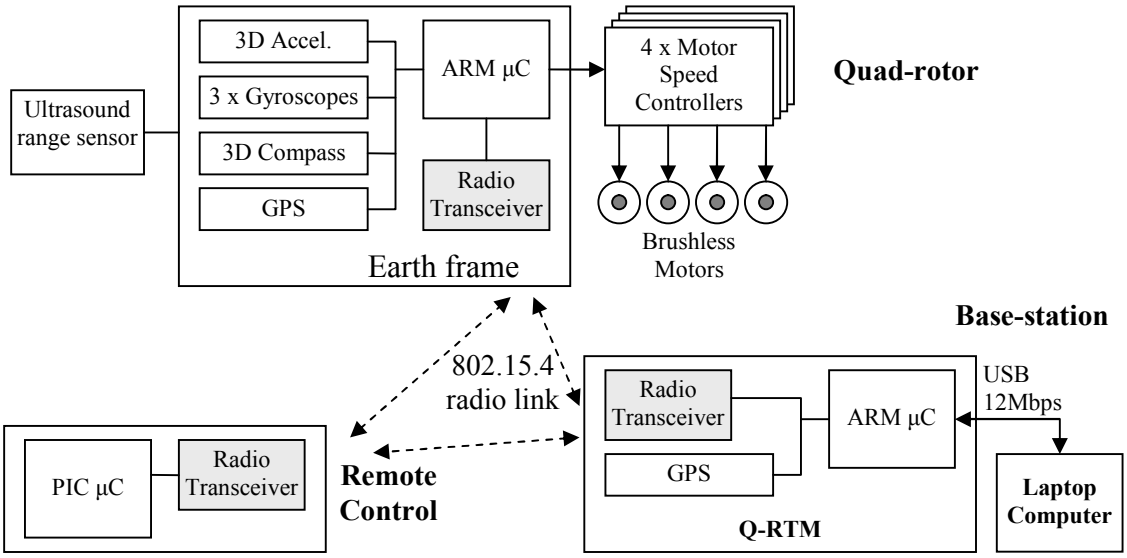


Figure 15. The electronic system for the quadrotor helicopter

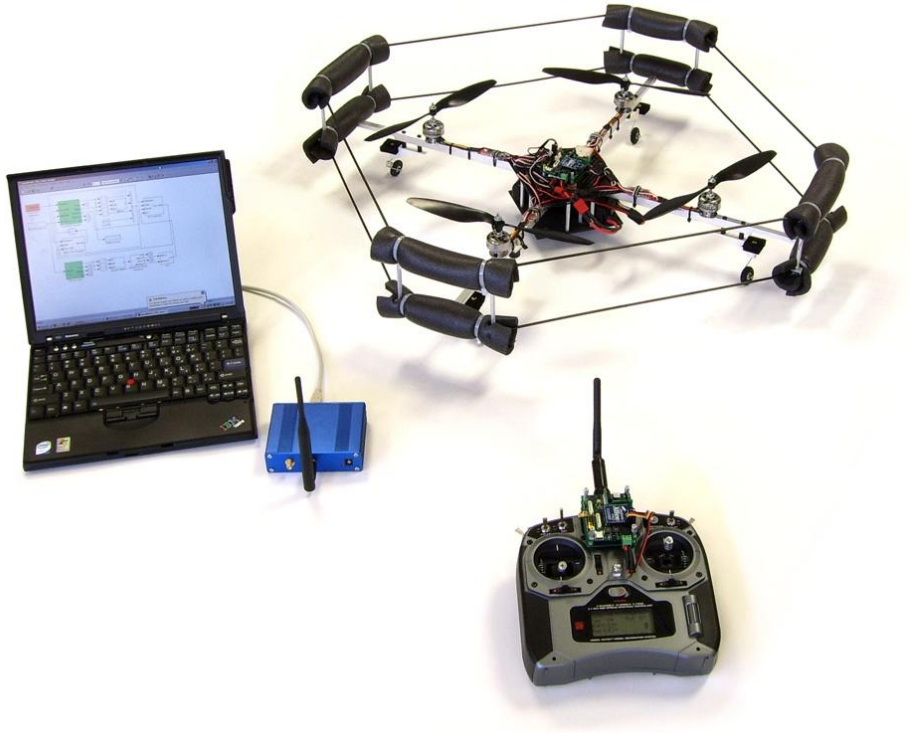


Figure 16. Quadrotor platform

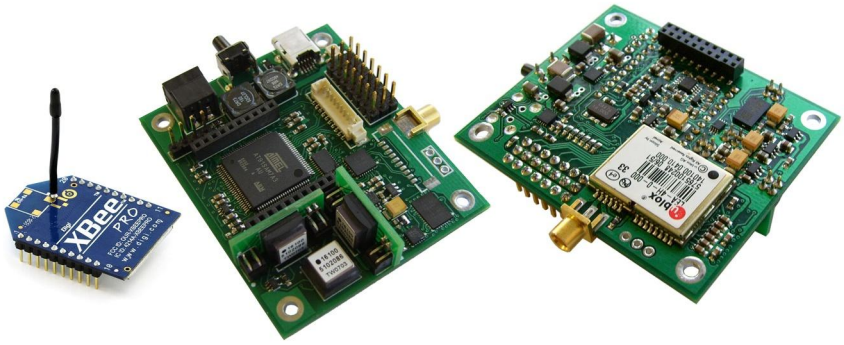


Figure 17. Quadrotor on-board autopilot

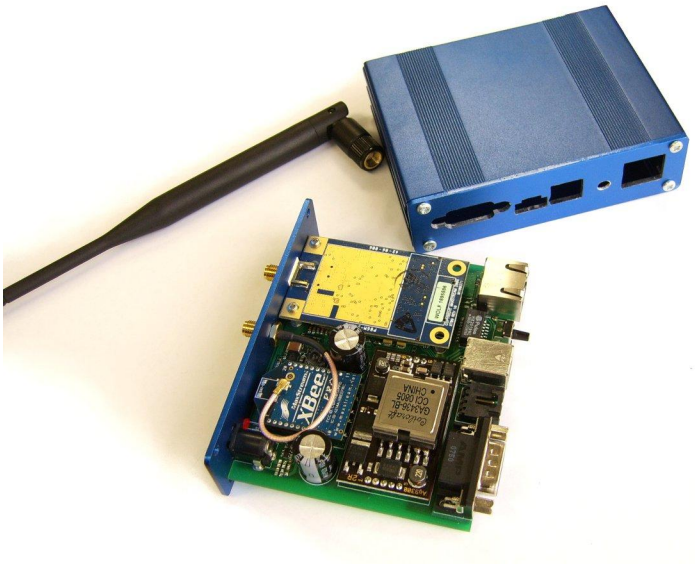


Figure 18. Basestation

The variables that can be measured by the on-board sensors are the following:

- angular velocities in body coordinates
- linear accelerations in the body coordinates
- the intensity of the Earth magnetic field in body coordinates
- position and velocities in Earth coordinates (GPS),
- the altitude using the ultrasound range sensor,

- temperatures of the gyroscopes, accelerometers and magnetic field sensors for calibration,
- battery voltage and current,
- motor rotation speeds.

3.2. Quadrotor Model

A quadrotor was selected as an implementation platform for the control algorithms. The quadrotor is unstable, and so makes an excellent platform for testing the suitability and effectiveness of control algorithms. Its dynamics are nonlinear, but relatively simple so that nonlinear control schemes can be designed and compared. Yet the quadrotor has enough nonlinearities to pose a meaningful challenge to controller design methods. Quadrotor is highly susceptible to wind gust disturbances, so that disturbance rejection capabilities of proposed controllers can be tested.

A good quadrotor model has to use theory usually applied for helicopters. Having four rotors in close proximity complicates the problem even further. There are interactions between the wakes produced by the rotors and the fuselage, and also between individual rotors. Except for hover, the expression for the rotor wash induced velocities cannot be obtained in closed-form, creating difficulties when the model is used to design certain types of controllers.

The derivation of the nonlinear dynamics is performed in the North-East-Down (NED) inertial coordinates and in the x-y-z body-fixed coordinates (Figure 19). Variables resolved to the inertial axes are denoted by an *e* subscript and the ones

resolved to the body axes have the *b* subscript. The attitude is represented using quaternions. Most equations are derived from [20].

The model has 17 states: body angular velocities, attitude quaternion, linear velocity, position, rotor speeds.

$$\mathbf{x} = \begin{bmatrix} \boldsymbol{\omega}_b & \mathbf{q} & \mathbf{V}_b & \mathbf{P}_e & \boldsymbol{\Omega} \end{bmatrix}^T \tag{14}$$

The inputs into the model are the four motor voltages:

$$\mathbf{u} = \mathbf{V}_{mot}$$

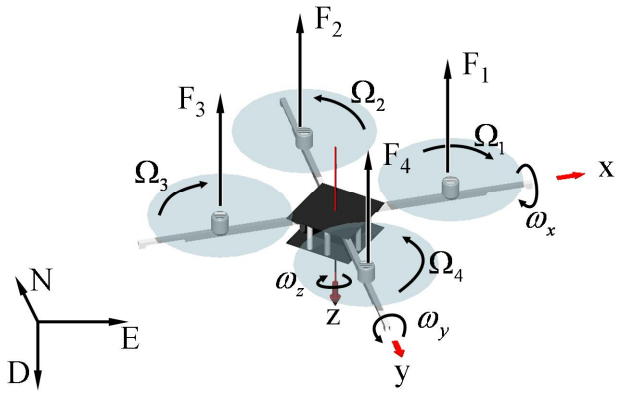


Figure 19. The axes definitions for the quadrotor model

Nomenclature

$\boldsymbol{\omega}_b$	angular velocity resolved to body frame (rad/s)
$\mathbf{q}$	attitude quaternion
$\mathbf{V}_b$	velocity in body frame (m/s)
$\mathbf{P}_e$	position in the inertial frame (m)
$\mathbf{I}_{nb}$	moment of inertia tensor $(kg \cdot m^2)$

$\mathbf{F}$	total force that acts at the center of gravity (N)
$\mathbf{M}$	total moment that acts at the center of gravity $(N \cdot m)$
m	total mass of the quadrotor (kg)
$F_1 - F_4$	thrust force for each rotor (N)
τ_{mot}, τ_{rot}	motor and rotor torques $(N \cdot m)$
$\mathbf{R}_{ROT-CG}$	vector of rotor placement relative to the center of mass (m)
Ω_i	rotation speed of the rotors (rad / s)
R	rotor radius (m)
b	number of blades for each rotor
I_{flap}	moment of inertia tensor for blade flapping $(kg \cdot m^2)$
$\mathbf{V}_{mot}$	motor voltages (V)
R_{mot}	winding resistance for the motors (Ω)
K_T, K_V	mechanical and electrical constants of the motors
ρ	air density (kg / m^3)
c	rotor blade chord (m)
a_0	linear lift-curve slope
C_L	lift coefficient
C_D	drag coefficient
d	rotor wash velocities (m / s)

Short definition of quaternions

The attitude is represented using quaternions. They parameterize the rotation from the inertial reference frame to the body frame using four values. The first is a scalar and the rest form a vector:

$$\mathbf{q} = \begin{bmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \end{bmatrix} = \begin{bmatrix} \cos(\alpha/2) \\ \sin(\alpha/2) \cdot \mathbf{r} \end{bmatrix}, \tag{15}$$

where α is the angle of rotation and $\mathbf{r}$ is the axis around which the rotation is made, with the three components resolved to the inertial axes. Rotation quaternions have unitary norm and the vector $\mathbf{r}$ is a unit vector also. Addition of rotation quaternions does not generally result in a rotation quaternion and has no physical meaning. A special non-commutative multiplication operation is defined for the quaternions, denoted with the $\otimes$ operator. Multiplying two or more rotation quaternions produces another rotation quaternion that represents the total rotation obtained by performing each individual rotation for each quaternion in reverse order, starting with the last term of the product. Vectors can be rotated from one axis system to another if they are first transformed in quaternions with a scalar part equal to zero. For example, the same vector $\mathbf{r}$ can be resolved to the inertial frame as $\mathbf{r}_e$ and to the body frame as $\mathbf{r}_b$. The relationship between $\mathbf{r}_e$ and $\mathbf{r}_b$ is the following:

$$\begin{bmatrix} 0 \\ \mathbf{r}_b \end{bmatrix} = \mathbf{q}^* \otimes \begin{bmatrix} 0 \\ \mathbf{r}_e \end{bmatrix} \otimes \mathbf{q}$$
$$\begin{bmatrix} 0 \\ \mathbf{r}_e \end{bmatrix} = \mathbf{q} \otimes \begin{bmatrix} 0 \\ \mathbf{r}_b \end{bmatrix} \otimes \mathbf{q}^* \tag{16}$$

where

$$\mathbf{q}^* = \mathbf{q}^{-1} = \begin{bmatrix} q_0 \\ -q_1 \\ -q_2 \\ -q_3 \end{bmatrix} \tag{17}$$

is the conjugate of the rotation quaternion. More details about quaternions can be found in [21]. To simplify things, an abuse of notation will be used for rotations of vectors using quaternions. Instead of writing the augmented version of the vector as a quaternion with a zero scalar part, the normal vector will be used instead. It will be considered that the result is the rotated vector.

Solid body dynamics

The kinematic and dynamic equations model the vehicle as a rigid body under the influence of the Earth gravity and the thrust forces produced by the rotors.

$$\dot{\boldsymbol{\omega}}_b = \mathbf{I}_{nb}^{-1} \left(\begin{bmatrix} M_x \\ M_y \\ M_z \end{bmatrix} - \boldsymbol{\omega}_b \times \left(\mathbf{I}_{nb} \boldsymbol{\omega}_b \right) \right) \tag{18}$$

$$\dot{\mathbf{q}} = \frac{1}{2} \mathbf{q} \otimes \bar{\boldsymbol{\omega}}_b \tag{19}$$

$$\dot{\mathbf{V}}_b = \frac{1}{m} \begin{bmatrix} F_x \\ F_y \\ F_z \end{bmatrix} + \mathbf{q}^* \otimes \begin{bmatrix} 0 \\ 0 \\ g \end{bmatrix} \otimes \mathbf{q} - \boldsymbol{\omega}_b \times \mathbf{V}_b \tag{20}$$

$$\dot{\mathbf{P}}_e = \mathbf{q} \otimes \mathbf{V}_b \otimes \mathbf{q}^* \tag{21}$$

$$\mathbf{M} = \mathbf{M}_{FUSE} + \sum_{i=1}^{4} \mathbf{M}_{Ri} \tag{22}$$

$$\mathbf{F} = \mathbf{F}_{FUSE} + \sum_{i=1}^{4} \mathbf{F}_{Ri} \tag{23}$$

where

$$\bar{\boldsymbol{\omega}}_b = \begin{bmatrix} 0 \\ \boldsymbol{\omega}_b \end{bmatrix} \tag{24}$$

Motor and rotor rotation dynamics

$$\tau_{mot\ i} = \frac{K_T}{R_{mot}} V_{mot\ i} - \frac{K_T}{K_V R_{mot}} \Omega_i \tag{25}$$

$$\dot{\Omega}_i = \frac{1}{bI_{flap}} \left(\tau_{mot\ i} - \tau_{rot\ i} \right) + \left(-1 \right)^i \dot{\omega}_{bz} \tag{26}$$

Rotor aerodynamics

The rotors are modeled as rigid propellers. Blade flapping and wake interaction are ignored. The *a* subscripts indicate aerodynamic velocities, *w* wash velocities and *i* inertial velocities.

$$\mathbf{V}_a = \begin{bmatrix} V_{xb} \\ \left(-1 \right)^{i+1} V_{yb} \\ V_{zb} \end{bmatrix} - \begin{bmatrix} V_{xw} \\ \left(-1 \right)^{i+1} V_{yw} \\ V_{zw} \end{bmatrix}, \quad i = 1..4 \tag{27}$$

$$\boldsymbol{\omega}_a = \begin{bmatrix} (-1)^{i+1}\omega_{xb} \\ \omega_{yb} \\ (-1)^{i+1}\omega_{zb} \end{bmatrix} - \begin{bmatrix} (-1)^{i+1}\omega_{xw} \\ \omega_{yw} \\ (-1)^{i+1}\omega_{zw} \end{bmatrix}, \quad i = 1..4 \tag{28}$$

$$\mathbf{V}_i = \begin{bmatrix} V_{xb} \\ (-1)^{i+1}V_{yb} \\ V_{zb} \end{bmatrix} - \begin{bmatrix} V_{xw} \\ (-1)^{i+1}V_{yw} \\ V_{zw} \end{bmatrix} + \begin{bmatrix} d_x \\ (-1)^{i+1}d_y \\ d_z \end{bmatrix}, \quad i = 1..4 \tag{29}$$

$$B_T = 1 - \frac{\sqrt{2|C_z|}}{b} \tag{30}$$

$$\sigma = \frac{bc}{\pi R} \tag{31}$$

$$\overline{C}_L = \frac{-6F_{za}}{\sigma} \tag{32}$$

$$C_D = \delta_0 + \delta_1 \overline{C}_L + \delta_2 \overline{C}_L^2 \tag{33}$$

$$\varepsilon_0 = \frac{C_D}{a_0} \tag{34}$$

$$\Omega_a = \Omega - \omega_{za} \tag{35}$$

$$V_{Ta} = \Omega_a R \tag{36}$$

The following variables are resolved in the wind-mast coordinates. This new reference system has the same vertical axis as the vehicle body, but it is rotated such that the new x coordinate points in the direction of the lateral aero velocity. All the aerodynamic expressions have a simpler form in this coordinate system.

$$V_{xa|WM} = \sqrt{V_{xa}^2 + V_{ya}^2} \tag{37}$$

The body angular velocities are also expressed in wind-mast coordinates:

$$\omega_{xa|WM} = \frac{V_{xa}}{\sqrt{V_{xa}^2 + V_{ya}^2}}\omega_{xa} + \frac{V_{ya}}{\sqrt{V_{xa}^2 + V_{ya}^2}}\omega_{ya}$$
$$\omega_{ya|WM} = -\frac{V_{ya}}{\sqrt{V_{xa}^2 + V_{ya}^2}}\omega_{xa} + \frac{V_{xa}}{\sqrt{V_{xa}^2 + V_{ya}^2}}\omega_{ya} \tag{38}$$

The non-dimensional velocities are defined as

$$\mu = \frac{V_{xa|WM}}{V_{Ta}}, \quad \lambda_a = \frac{V_{za}}{V_{Ta}}, \quad \lambda_w = \frac{V_{zw}}{V_{Ta}}, \quad \lambda_i = \frac{V_{zi}}{V_{Ta}} \tag{39}$$

$$\hat{p}_a = \frac{\omega_{xa|WM}}{\Omega_a}, \quad \hat{q}_a = \frac{\omega_{ya|WM}}{\Omega_a}. \tag{40}$$

From the following expressions for the force and torque coefficients it can be seen that they strongly depend on the vertical (λ) and the longitudinal (μ) velocities. This creates a strong coupling between the inertial dynamics and the aerodynamics.

$$C_{x|WM} = \frac{\sigma a_0}{2}\left[\begin{array}{l} -\left(\frac{B_T^2\mu}{2}\right)\varepsilon_0 + \left(\frac{B_T\mu}{2}\lambda_a\right)\theta_0 + \left(\frac{B_T^2\mu}{4}\lambda_a\right)\theta_T + \\ +\left(\frac{B_T^3}{6}\theta_0 + \frac{B_T^4}{8}\theta_T + \frac{B_T^2}{2}\lambda_a\right)\hat{p}_a \end{array}\right] \tag{41}$$

$$C_{y|WM} = \frac{\sigma a_0}{2}\left[\left(\frac{B_T^3}{6}\theta_0 + \frac{B_T^4}{8}\theta_T + \frac{B_T^2}{2}\lambda_a\right)\hat{q}_a\right] \tag{42}$$

The following three equations may create an implicit dependence between the thrust force and the self-induced wash velocity. A dynamic model of the rotor wake is used. Normally the wake has its own dynamics and a good model has to consider it, but this adds extra states to the already complicated model, and even worse these states can not be measured in real applications. For precise simulations the dynamics can be kept and updated along with the main model. One of the advantages of the extra states is that

the relation between the thrust force and the wash velocity becomes explicit. For normal modeling it is simpler to iterate locally through equations (43)-(45) until steady-state is reached and use those values for C_z and λ_w for the rest of the model.

$$C_z = -\frac{\sigma a_0}{2}\left[\begin{array}{l}\left(\frac{B_T^3}{3}+\frac{\mu^2 B_T}{2}\right)\theta_0+\left(\frac{B_T^4}{4}+\frac{\mu^2 B_T^2}{4}\right)\theta_T+\\ +\left(\frac{B_T^2}{2}+\frac{\mu^2}{4}\right)(1+\varepsilon_0)\lambda_a+\left(\frac{B_T^2\mu}{4}\right)(1+\varepsilon_0)\hat{p}_a\end{array}\right] \tag{43}$$

$$\frac{4k^3}{3}\dot{\lambda}_w = -2\sqrt{\mu^2+\lambda_a^2}\lambda_w - C_z \tag{44}$$

The dimensionless aero velocity takes the self-induced rotor wash into account:

$$\lambda_a = \lambda_i - \lambda_w \tag{45}$$

$$C_{mx|WM} = \frac{\sigma a_0}{2}\left[-\left(\frac{B_T^3\mu}{3}\right)\theta_0-\left(\frac{B_T^4\mu}{4}\right)\theta_T-\left(\frac{B_T^2\mu}{4}\right)\lambda_a-\left(\frac{B_T^4}{8}\right)\hat{p}_a\right] \tag{46}$$

$$C_{my|WM} = \frac{\sigma a_0}{2}\left[-\left(\frac{B_T^4}{8}\right)\hat{q}_a\right] \tag{47}$$

$$C_q = \frac{\sigma a_0}{2}\left[\begin{array}{l}\left(\frac{B_T^4}{4}+\frac{B_T^2\mu^2}{4}\right)\varepsilon_0-\left[\left(\frac{B_T^3}{3}\right)\theta_0+\left(\frac{B_T^4}{4}\right)\theta_T+\left(\frac{B_T^2}{2}\right)\lambda_a\right]\lambda_a+\\ +\left(-\frac{B_T^3\mu}{6}\theta_0-\frac{B_T^4\mu}{8}\theta_T-\frac{B_T^4}{8}\hat{p}_a\right)\hat{p}_a+\left(-\frac{B_T^4}{8}\hat{q}_a\right)\hat{q}_a\end{array}\right] \tag{48}$$

$$\tau_{rot} = \rho\pi R^3 V_{Ta}^2 C_q \tag{49}$$

The x and y force and moment coefficients are resolved back to the body axes:

$$C_x = \frac{V_{xa}}{\sqrt{V_{xa}^2+V_{ya}^2}}C_{x|WM} - \frac{V_{ya}}{\sqrt{V_{xa}^2+V_{ya}^2}}C_{y|WM} \tag{50}$$

$$C_y = \frac{V_{ya}}{\sqrt{V_{xa}^2 + V_{ya}^2}} C_{x|WM} + \frac{V_{xa}}{\sqrt{V_{xa}^2 + V_{ya}^2}} C_{y|WM} \tag{51}$$

$$C_{mx} = \frac{V_{xa}}{\sqrt{V_{xa}^2 + V_{ya}^2}} C_{mx|WM} - \frac{V_{ya}}{\sqrt{V_{xa}^2 + V_{ya}^2}} C_{my|WM} \tag{52}$$

$$C_{my} = \frac{V_{ya}}{\sqrt{V_{xa}^2 + V_{ya}^2}} C_{mx|WM} + \frac{V_{xa}}{\sqrt{V_{xa}^2 + V_{ya}^2}} C_{my|WM} \tag{53}$$

Rotor forces and moments at the C.G.

$$\mathbf{F}_R = \rho \pi R^2 V_{Ta}^2 \begin{bmatrix} C_x \\ (-1)^i C_y \\ C_z \end{bmatrix} \tag{54}$$

$$\mathbf{M}_R = \begin{bmatrix} 0 \\ 0 \\ (-1)^i \tau_{mot} \end{bmatrix} + \rho \pi R^3 V_{Ta}^2 \begin{bmatrix} (-1)^i C_{mx} \\ C_{my} \\ 0 \end{bmatrix} + b I_{flap} \Omega \begin{bmatrix} (-1)^i \omega_{yb} \\ \omega_{xb} \\ 0 \end{bmatrix} + \mathbf{R}_{ROT-CG} \times \mathbf{F}_R \tag{55}$$

Rotor self-induced velocity and wash interaction

The Glauert model is used for the propellers seen as actuator disks. The induced wash velocity depends on the thrust force:

$$\mathbf{d} = \frac{-1}{2\rho |\mathbf{V}_a|} \begin{bmatrix} 1/\pi R^2 & 0 & 0 \\ 0 & 1/\pi R^2 & 0 \\ 0 & 0 & 1/\pi R^2 \end{bmatrix} \mathbf{F}_R \tag{56}$$

The only place where wind appears in the model is here. Wind is included in the wash velocity after it is resolved to body coordinates. Wash velocity affects the aerodynamic velocity, so wind appears in most aerodynamic equations implicitly.

$$\mathbf{V}_w = \mathbf{d} + \mathbf{q}^* \otimes \mathbf{V}_{wind} \otimes \mathbf{q} \tag{57}$$

There is no rotating wake modeled.

$$\boldsymbol{\omega}_w = \begin{bmatrix} 0 & 0 & 0 \end{bmatrix}^T \tag{58}$$

The variable i is odd for clockwise rotor rotation and even for anti-clockwise rotation; τ_{mot}, τ_{rot}, Ω are always positive and independent of the direction of rotation.

3.3. Dynamic Inversion of the Quadrotor Model

The dynamic representation of the quadrotor system can be written as

$$\dot{\mathbf{x}} = \begin{bmatrix} \dot{\boldsymbol{\omega}}_b \\ \dot{\mathbf{q}} \\ \dot{\mathbf{V}}_b \\ \dot{\mathbf{P}}_e \\ \dot{\boldsymbol{\Omega}} \end{bmatrix} = \begin{bmatrix} f_\omega(\boldsymbol{\omega}_b, \mathbf{q}, \boldsymbol{\Omega}) \\ f_q(\boldsymbol{\omega}_b, \mathbf{q}) \\ f_V(\boldsymbol{\omega}_b, \mathbf{q}, \mathbf{V}_b) \\ f_P(\mathbf{q}, \mathbf{V}_b) \\ f_\Omega(\boldsymbol{\Omega}, \boldsymbol{\omega}_b, \mathbf{q}, \mathbf{V}_b, \mathbf{V}_{mot}) \end{bmatrix} \tag{59}$$

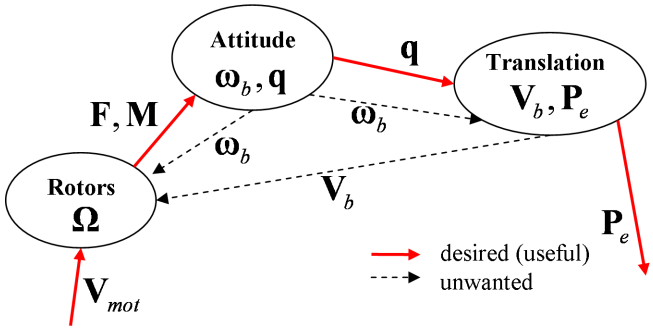


Figure 20. The three subsystems of the quadrotor model

From the equations of the quadrotor model it is possible to obtain some approximate inverse functions for f_ω, f_V and f_Ω. These functions are of interest:

$$\begin{aligned}\mathbf{V}_{mot} &= f_{\Omega}^{-1}\left(\dot{\mathbf{\Omega}}_{des}, \boldsymbol{\omega}_b, \mathbf{q}, \mathbf{V}_b\right) \\ \mathbf{\Omega} &= f_{\omega}^{-1}\left(\dot{\boldsymbol{\omega}}_{b\,des}, \mathbf{q}\right) \\ \mathbf{q} &= f_{V}^{-1}\left(\boldsymbol{\omega}_b, \dot{\mathbf{V}}_{b\,des}\right)\end{aligned} \tag{60}$$

The following steps are necessary in order to obtain the inverse functions:

1. Solve for F_z and q given a_N, a_E, a_D and Ψ:

Condition for having F_z parallel to the acceleration vector: $F_x = 0, F_y = 0$.

The effect of the $\boldsymbol{\omega}_b \times \mathbf{V}_b$ term is ignored.

$$\dot{\mathbf{V}}_b = \frac{1}{m}\begin{bmatrix}0\\0\\F_z\end{bmatrix} + \mathbf{q}^* \otimes \begin{bmatrix}0\\0\\g\end{bmatrix} \otimes \mathbf{q} = \mathbf{q}^* \otimes \begin{bmatrix}a_N\\a_E\\a_D\end{bmatrix} \otimes \mathbf{q} \tag{61}$$

$$\frac{1}{m}\begin{bmatrix}0\\0\\F_z\end{bmatrix} = \mathbf{q}^* \otimes \begin{bmatrix}a_N\\a_E\\a_D - g\end{bmatrix} \otimes \mathbf{q} \tag{62}$$

$$F_z = m\sqrt{a_N^2 + a_E^2 + \left(a_D - g\right)^2} \tag{63}$$

The necessary attitude $\mathbf{q}$ is obtained by calculating the yaw and the tilt configurations separately as rotations around the vertical (D) axis and around an axis in the horizontal plane that will make the *z* axis parallel to the desired acceleration vector:

$$\mathbf{q}_{\Psi} = \begin{bmatrix}\cos\frac{\Psi}{2}\\ \sin\frac{\Psi}{2}\begin{bmatrix}0\\0\\1\end{bmatrix}\end{bmatrix} = \begin{bmatrix}\cos\frac{\Psi}{2}\\0\\0\\\sin\frac{\Psi}{2}\end{bmatrix} \tag{64}$$

$$\mathbf{r} = \begin{bmatrix} 0 \\ 0 \\ -1 \end{bmatrix} \times \begin{bmatrix} a_N \\ a_E \\ a_D - g \end{bmatrix} \tag{65}$$

$$\alpha = \arctan_2 \left(\frac{\sqrt{a_N^2 + a_E^2}}{-(a_D - g)} \right) \tag{66}$$

$$\mathbf{q}_\alpha = \begin{bmatrix} \cos\dfrac{\alpha}{2} \\ \sin\dfrac{\alpha}{2} \cdot \dfrac{\mathbf{r}}{|\mathbf{r}|} \end{bmatrix} \tag{67}$$

$$\mathbf{q} = \mathbf{q}_\alpha \otimes \mathbf{q}_\Psi \tag{68}$$

2. Solve for $\Omega_1, \Omega_2, \Omega_3, \Omega_4$ given $\dot{\omega}_x, \dot{\omega}_y, \dot{\omega}_z, \dot{V}_z$:

There is no closed-form solution to this problem because the thrust force and the induced velocity of the rotors depend on each other in equations (43)-(45). A Newton-Raphson algorithm is used to trim the quadrotor model for the desired accelerations and to obtain the necessary rotor speeds. To calculate a starting value for Ω, hover conditions are assumed (i.e. no horizontal or vertical speed). Some terms cancel in equation (43) and for simplicity $B_T = 1$ and $\varepsilon = 0$. Using the Glauert momentum model the relation between the induced velocity and thrust is

$$w = \frac{T}{2\rho A \|V\|}. \tag{69}$$

Because at hover the total velocity $V = w$ and $F_z = -T$ (by definition the thrust T points upwards, but F_z is positive downwards) it is easy to see that

$$w_{hover} = \sqrt{\frac{-F_z}{2\rho A}} \tag{70}$$

and λ_a can now be calculated from w_{hover}. In (43) Ω_{start} is the solution of a second-order equation that depends on F_z. The acceleration $\dot{V}_z$ includes the effect of g (i.e. $\dot{V}_z = 0$ means hover) so

$$F_z = \min\left\{(\dot{V}_z - g_z)\frac{m}{4}, 0\right\} \tag{71}$$

where g_z is the projection of the gravitational acceleration on the *z* axis:

$$g_z = \begin{bmatrix}0 & 0 & 1\end{bmatrix}\left(\mathbf{q}^* \otimes \begin{bmatrix}0\\0\\g\end{bmatrix} \otimes \mathbf{q}\right) \tag{72}$$

The general form of the quadrotor model is

$$\dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{u}). \tag{73}$$

During the Newton-Raphson iterations, all the states $\mathbf{x}$ are kept constant and equal to their values at the current operating point, except for the states $\Omega_1, \Omega_2, \Omega_3, \Omega_4$ which are used as inputs. Only the $\dot{\omega}_x, \dot{\omega}_y, \dot{\omega}_z, \dot{V}_z$ components of the function f are considered as outputs. Instead of running the full model, from equation (59) it can be seen that f_ω and f_V are sufficient. Numerical perturbation on the Ω inputs is used to construct the Jacobian matrix J. The solution for $\mathbf{\Omega} = \begin{bmatrix}\Omega_1 & \Omega_2 & \Omega_3 & \Omega_4\end{bmatrix}^T$ is found after a few iterations:

$$J_k = \begin{bmatrix} \frac{\partial f_{\omega x}}{\partial \Omega_1} & \cdots & \cdots & \frac{\partial f_{\omega x}}{\partial \Omega_4} \\ \frac{\partial f_{\omega y}}{\partial \Omega_1} & & & \frac{\partial f_{\omega y}}{\partial \Omega_4} \\ \frac{\partial f_{\omega z}}{\partial \Omega_1} & & & \frac{\partial f_{\omega z}}{\partial \Omega_4} \\ \frac{\partial f_{Vz}}{\partial \Omega_1} & \cdots & \cdots & \frac{\partial f_{Vz}}{\partial \Omega_4} \end{bmatrix}_{[\Omega_{1..4}]=\mathbf{\Omega}_k} \tag{74}$$

and

$$\mathbf{\Omega}_{k+1} = \mathbf{\Omega}_k - J_k^{-1} \left(\begin{bmatrix} \dot{\omega}_{x\,k} \\ \dot{\omega}_{y\,k} \\ \dot{\omega}_{z\,k} \\ \dot{V}_{z\,k} \end{bmatrix} - \begin{bmatrix} \dot{\omega}_{x\,ref} \\ \dot{\omega}_{y\,ref} \\ \dot{\omega}_{z\,ref} \\ \dot{V}_{z\,ref} \end{bmatrix} \right) \tag{75}$$

The inverse of J_k can not be calculated when the rotors are in vortex ring state. The model is not valid for high descending velocities, comparable to the rotor wash velocity.

3. Solve for $V_{mot1}, V_{mot2}, V_{mot3}, V_{mot4}$ given $\dot{\Omega}_1, \dot{\Omega}_2, \dot{\Omega}_3, \dot{\Omega}_4$:

The following equations are written without the motor index for simplicity.

The electro-mechanical system is described by

$$\tau_{mot} = \frac{K_T}{R} V_{mot} - \frac{K_T}{K_V R} \Omega \tag{76}$$

$$V_{mot} = \frac{R}{K_T} \tau_{mot} + \frac{1}{K_V} \Omega \tag{77}$$

The inertial dynamics of the motor and propeller is

$$\dot{\Omega} = \frac{1}{bI_{flap}} \left(\tau_{mot} - \tau_{rot} \right) + \left(-1 \right)^i \dot{\omega}_{bz} \tag{78}$$

$$\tau_{mot} = bI_{flap}\dot{\Omega} + \tau_{rot} - bI_{flap}\left(-1\right)^{i}\dot{\omega}_{bz} \tag{79}$$

Finally the motor voltage necessary to achieve a certain rotation acceleration is

$$V_{mot} = \frac{R}{K_T}\left[bI_{flap}\dot{\Omega} + \tau_{rot} - bI_{flap}\left(-1\right)^{i}\dot{\omega}_{bz}\right] + \frac{1}{K_V}\Omega \tag{80}$$

3.4. Hardware Implementation of the Ground Robots

Introduction

Motion path planning is the process of finding a continuous path from an initial position to a prescribed final position (goal) without collision. Artificial potential field methods for obstacle avoidance have gained increased popularity among researchers in the field of mobile robots. It is generally easy to develop centralized algorithms that apply potential field methods for all robots based on complete information about their state variables. For many applications, distributed control algorithms are required to operate on each mobile robot under conditions of restricted communication channels between the robots. The ground robots testbed allows for testing of these algorithms for a formation of mobile robots that communicate over a graph structure.

Mobile Robots Testbed

The testbed for the experiments consists in a set of different mobile robot platforms with the model from Figure 23 but with different parameters that result in different dynamics. Currently there are three types of robots as shown in Figure 21.

Figure 21. Different types of mobile robots

Each robot has on-board electronics that provides radio communication, controls the linear velocity and the heading rate using PID controllers and measures the linear acceleration, the wheels speed and the heading rate. Each robot can receive the sensor information from any other robot by radio. The same information can optionally be received by a computer that can implement centralized control or can simulate distributed control (Figure 22). The graph structure that describes the communication links between the robots can be the one dictated by the actual communication conditions where some robots are too far away from others and can not maintain a reliable link with all the members of the formation, or it can be manually set to a

specific configuration when the formation is not spread out too far and the communication is done reliably between all the members of the formation. The sample rate is 20 Hz. For ease of implementation and for testing various graph configurations, the formation is controlled by a computer that holds all the state information from all the robots but simulates local, distributed control algorithms.

The sensors present on the robots do not provide absolute position information. A video camera is used for absolute localization. Each robot is equipped with a high-intensity LED. The frame rate of the camera is synchronized with the sample rate of the robot sensors and of their actuators. Each LED is activated one at a time for a sample period. At each frame, the camera captures only one LED. The computer calculates the position of the brightest point in each video frame and based on this information the position of each robot within the testbed is obtained.

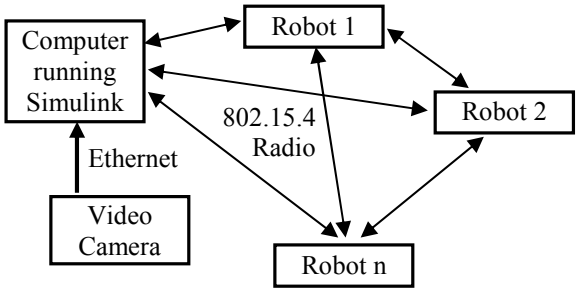


Figure 22. Communication topology.

3.5. Ground Robots Model

Nomenclature

Ω_L, Ω_R	angular velocities of the wheels (rad / s)
d	wheels diameter (m)
l	distance between wheels (m)
x, y	position coordinates in the inertial frame (m)
θ	heading angle (rad)
V	velocity in the body frame (m / s)
ω	angular velocity (rad / s)
a	acceleration in the body frame (m / s^2)

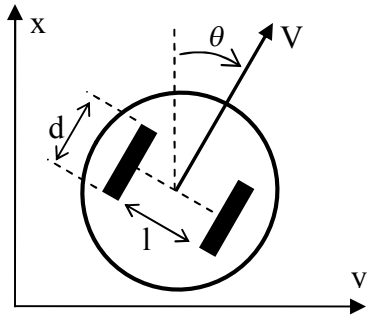


Figure 23. Mobile ground robot model.

The robots (Figure 23) have two wheels powered by individual motors and optionally a caster wheel. The dynamics is described by the following equations:

$$\begin{aligned} \dot{x} &= V\cos\theta \\ \dot{y} &= V\sin\theta \\ \dot{V} &= a \\ \dot{\theta} &= \omega \end{aligned} \tag{81}$$

The robots provide direct measurements for a using accelerometers, ω using gyroscopes, and indirect measurements for V and ω using wheel encoders:

$$\begin{aligned} V &= \frac{d}{2}\left(\Omega_L + \Omega_R\right) \\ \omega &= \frac{d}{l}\left(\Omega_L - \Omega_R\right) \end{aligned} . \tag{82}$$

The x and y measurements are provided by the video camera system for each robot.

Kalman Filter

A discrete-continuous extended Kalman filter (EKF) is used to estimate the states of each robot [19]. The model used for the system is

$$\dot{x}(t) = f\left(x(t), u(t)\right) + G(t)w(t), \quad w(t) \sim N\left(0, Q(t)\right) \tag{83}$$

$$\tilde{y}_k = h(x_k) + v_k, \quad v_k \sim N\left(0, R_k\right). \tag{84}$$

The initialization is done as follows:

$$\hat{x}(t_0) = \hat{x}_0 \tag{85}$$

$$P_0 = E\left\{x(t_0)x^T(t_0)\right\} \tag{86}$$

The gain-update equations are

$$K_k = P_k^- H_k^T(\hat{x}_k^-)\left[H_k(\hat{x}_k^-)P_k^- H_k^T(\hat{x}_k^-) + R_k\right]^{-1} \tag{87}$$

where

$$H_k(\hat{x}_k^-) = \left.\frac{\partial h}{\partial x}\right|_{\hat{x}_k^-} \tag{88}$$

and

$$x_k^+ = x_k^- + K_k\left[\tilde{y}_k - h(\hat{x}_k^-)\right] \tag{89}$$

$$P_k^+ = \left[I - K_k H_k(\hat{x}_k^-)\right]P_k^- \tag{90}$$

The time propagation equations are

$$\dot{\hat{x}}(t) = f(\hat{x}(t), u_k) \tag{91}$$

$$\dot{P}(t) = F\left(\hat{x}(t), t\right)P(t) + P(t)F^T\left(\hat{x}(t), t\right) + G(t)Q(t)G^T(t) \tag{92}$$

where

$$F\left(\hat{x}(t), t\right) = \left.\frac{\partial f}{\partial x}\right|_{\hat{x}(t)}. \tag{93}$$

The state vector used in the Kalman filter for each robot has the following components:

$$\mathbf{x} = \begin{bmatrix} x & y & \theta & V & \omega \end{bmatrix}^T. \tag{94}$$

The inputs $u(t)$ used for the time propagation are the acceleration a measured by the robots and a value of zero for $\dot{\omega}$. We thus have

$$f(\mathbf{x}) = \begin{bmatrix} V\cos\theta \\ V\sin\theta \\ \omega \\ a \\ 0 \end{bmatrix}. \tag{95}$$

The process noise covariance matrix is dependent on the state variable θ:

$$Q(t) = diag\begin{bmatrix} q_V\cos(\theta) & q_V\sin(\theta) & q_\omega & q_a & q_{\dot{\omega}} \end{bmatrix}. \tag{96}$$

The available state measurements are

$$y_k = \begin{bmatrix} x & y & \Omega_L & \Omega_R & \omega \end{bmatrix}^T \tag{97}$$

and the measurement model is

$$h(x_k) = \begin{bmatrix} x \\ y \\ \dfrac{2V + l\omega}{2d} \\ \dfrac{2V - l\omega}{2d} \\ \omega \end{bmatrix}. \tag{98}$$

CHAPTER 4

STRUCTURED FLIGHT CONTROLLER FOR A QUADROTOR

4.1. Nomenclature

$\boldsymbol{\omega}_b$	angular velocity resolved to body frame (rad/s)
$\mathbf{q}$	attitude quaternion
$\mathbf{V}_b$	velocity in body frame (m/s)
$\mathbf{R}_e$	position in the inertial frame (m)
$\mathbf{I}_{nb}$	moment of inertia tensor $(kg \cdot m^2)$
F_z	total force of the rotors on the z axis (N)
$M_x - M_z$	total rotor moments along each axis $(N \cdot m)$
$F_1 - F_4$	thrust force for each rotor (N)
Q	rotor torque $(N \cdot m)$
T	rotor thrust force (N)
D	rotor diameter (m)
d	offset of each rotor from the center of mass (m)
n_i	rotation frequency of the rotors (Hz)
Ω_i	rotation speed of the rotors (rad/s)

4.2. Introduction

Quadrotor helicopters have become popular for research in UAV control due to their relatively simple model and the low-cost involved in operating the experimental platforms. Many groups were successful in developing autonomous quadrotor vehicles. Until only a few years ago, good results were obtained exclusively by using tethers or motion guides, or by having precise external sensors to track the attitude and position [22–24]. Today there are a few projects that are able to do autonomous indoor or outdoor flight using only on-board sensors for attitude estimation and without needing any motion-constraining device. The project in [25] uses a commercially-available remote-controlled vehicle on which an inertial measurement unit (IMU) and a digital signal processing board were installed. Only the attitude is controlled as there is no mechanism to measure the position of the quad-rotor. Quaternion representation is used for the attitude and backstepping techniques are applied to drive the error quaternion to zero. The OS4 project [26] uses integral backstepping for full control of attitude, altitude and position on a custom platform. The attitude is sensed using an IMU, the altitude using an ultrasound range sensor and the horizontal position using an external vision system. Finally, the vehicle developed by the STARMAC project [27] has the ability to fly outdoor. A comprehensive model was developed that includes the induced air velocity and the effects of blade flapping in translational flight. Unlike the OS4, a precise differential GPS unit can be optionally used as a replacement of the vision system to measure the position and the linear velocities.

This chapter presents a structured flight controller for the quadrotor platform developed at the Automation & Robotics Research Institute. A simplified model is developed and its parameters are identified. It is used to design a proportional controller for stabilizing the attitude and the altitude. Quaternion representation is employed.

The testbed for the experiments consists in a quadrotor platform with on-board sensors and brushless motors, a ground computer and a remote control. The control algorithms are implemented in Simulink (Figure 24) on the ground computer and run directly in the Matlab environment in normal or accelerated mode, allowing an instant transition between design and experiments.

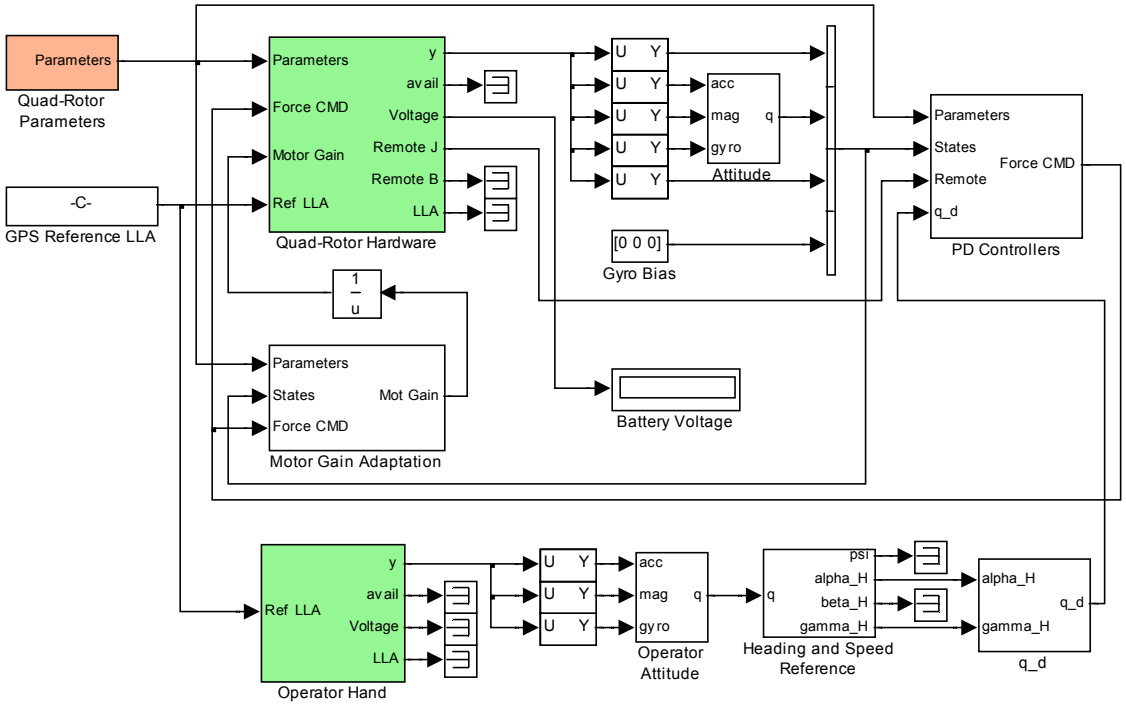


Figure 24. Top-level Simulink implementation of the control algorithm

Instead of the joysticks on the remote control, a pilot can use a second quad-rotor electronic board equipped with sensors to control the vehicle in a more intuitive

manner. By tilting the board in different directions the pilot gives a vector velocity reference to the quadrotor. The compass is used on both the pilot board and the vehicle board such that independent of the yaw angle, the quadrotor follows the commands given by the pilot in the correct direction in the inertial reference frame. This greatly simplifies the work of the pilot and allows persons that have no previous training to control the vehicle in a natural way.

4.3. Simplified Quadrotor Model

Modeling quadrotors is not an easy task. A good model has to use theory usually applied for helicopters. Having four rotors in close proximity complicates the problem even further. There are interactions between the wakes produced by the rotors and the fuselage, and also between individual rotors. Because the propellers are made of plastic, they are quite flexible and present flapping at translational speeds. They can not be modeled precisely as propellers and require models similar to helicopter rotors. Except for hover, the expression for the rotor wash induced velocities can not be obtained in closed-form, creating difficulties when the model is used to design certain types of controllers.

The approach taken in this chapter is to model only the most important elements of the quadrotor that define its behavior at hover and ignore the ones that have a significant effect only at high speeds.

The derivation of the nonlinear dynamics is performed in the North-East-Down (NED) inertial coordinates and in the x-y-z body-fixed coordinates (Figure 25).

Variables resolved to the inertial axes will be denoted by an *e* subscript and the ones resolved to the body axes will have the *b* subscript. The attitude is represented using quaternions.

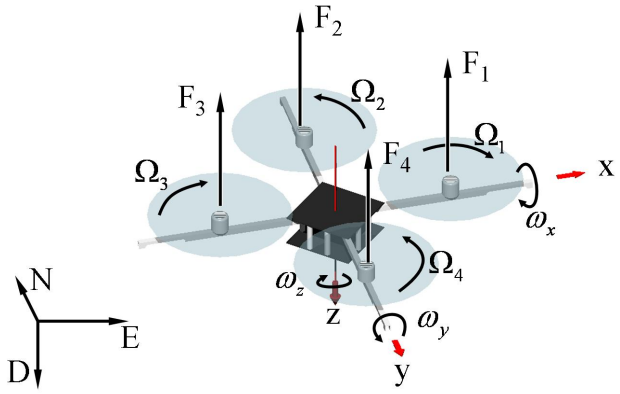


Figure 25. Quadrotor model

The kinematic and dynamic equations model the vehicle as a rigid body under the influence of the Earth gravity and the thrust forces produced by the rotors. Blade flapping, wake interaction and any other effects caused by the translational velocity are ignored.

$$\dot{\mathbf{V}}_b = \frac{1}{m}\begin{bmatrix} 0 \\ 0 \\ F_z \end{bmatrix} + \mathbf{q}^* \otimes \begin{bmatrix} 0 \\ 0 \\ g \end{bmatrix} \otimes \mathbf{q} - \boldsymbol{\omega}_b \times \mathbf{V}_b \tag{99}$$

$$\dot{\boldsymbol{\omega}}_b = \mathbf{I}_{nb}^{-1}\left(\begin{bmatrix} M_x \\ M_y \\ M_z \end{bmatrix} - \boldsymbol{\omega}_b \times \left(\mathbf{I}_{nb}\boldsymbol{\omega}_b\right)\right) \tag{100}$$

$$\dot{\mathbf{R}}_e = \mathbf{q} \otimes \mathbf{V}_b \otimes \mathbf{q}^* \tag{101}$$

$$\dot{\mathbf{q}} = \frac{1}{2}\mathbf{q} \otimes \bar{\boldsymbol{\omega}}_b, \tag{102}$$

where

$$\bar{\boldsymbol{\omega}}_b = \begin{bmatrix} 0 \\ \boldsymbol{\omega}_b \end{bmatrix}. \tag{103}$$

The rotors are modeled as propellers in hover. The gyroscopic effect is ignored because the mass of all four propellers is only about 3% of the total mass of the vehicle. The thrust coefficient K_T and the torque coefficient K_Q were identified by measuring the forces and moments generated by the motors for given rotation speeds:

$$K_T = \frac{T}{\rho n^2 D^4} \tag{104}$$

$$K_Q = \frac{Q}{\rho n^2 D^5}. \tag{105}$$

The rotation velocities of the rotors are taken in their absolute value and are always positive. The direction of rotation is accounted for in the matrix in equation (108).

$$\Omega_i = 2\pi n_i \tag{106}$$

$$\tau\dot{\Omega}_i + \Omega_i = a_i f_{mot}\left(u_i, V_{batt}\right), \quad i = \overline{1,4} \tag{107}$$

Equation (107) models the behavior of the brushless DC motors with the propellers attached. From experiments it was observed that the transfer function of the motors together with the speed controllers is of first order. The time constant τ is 0.08 seconds, ensuring a high enough bandwidth for proper control. The input to the speed controllers is a servo-type PWM signal where the command u_i is sent as a variable-length rectangular pulse. The speed response of the motor is not linear and is

approximated by the f_{mot} function that was identified experimentally. The battery voltage is measured online. To take into account the slightly different behavior of each motor, a gain a_i close to one was included also. This gain is estimated during flight for each motor and the suitable command is calculated in order to cancel its effect and to obtain an identical behavior for all motors.

The controllers used to stabilize the quadrotor generate a desired vertical force and desired moments about each axis in the body frame. The equation below maps the forces that are generated by each rotor to the vertical force and the moments. The controller will use the inverse of the matrix in equation (108) to obtain the rotor forces as a function of the commanded vertical force and moments:

$$\begin{bmatrix} F_z \\ M_x \\ M_y \\ M_z \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & -d & 0 & d \\ -d & 0 & d & 0 \\ c & -c & c & -c \end{bmatrix} \cdot \begin{bmatrix} F_1 \\ F_2 \\ F_3 \\ F_4 \end{bmatrix} \tag{108}$$

where

$$c = \frac{K_Q D}{K_T}. \tag{109}$$

The inputs into the quadrotor model are the desired forces $F_{di}, i = \overline{1,4}$. The actual platform receives a servo-type PWM signal u_i as an input. Equation (110) calculates the desired rotor speed and equation (111) the corresponding command u_i:

$$\Omega_{di} = \frac{2\pi}{D^2}\sqrt{\frac{-F_{di}}{K_T \rho}} \tag{110}$$

$$u_i = f_{mot}^{-1}\left(\frac{1}{a_i}\Omega_{di}, V_{batt}\right) \tag{111}$$

4.4. Attitude Estimation

The attitude of the quadrotor is estimated using the readings from the on-board inertial and magnetic sensors in body coordinates. The angular velocity $\boldsymbol{\omega}_b$ read using the gyroscopes is integrated to produce a high-bandwidth, low-noise estimate of the attitude quaternion (Figure 26). All sensors are temperature compensated and show no significant bias. Still, in the case of the gyroscopes, even small biases below the noise floor can cause significant drift if integrated. The accelerometer and the magnetometer readings are used together as inputs to the QUEST algorithm [28] to estimate the attitude quaternion also. In this case, because of vibrations and the body acceleration, the estimate is noisy. In the long term, it does not drift because it is based on absolute measurements. That's why it is used to do slow corrections to the estimate from the gyroscopes. This way, the final estimate is almost free of noise, has high bandwidth and presents no significant drift.

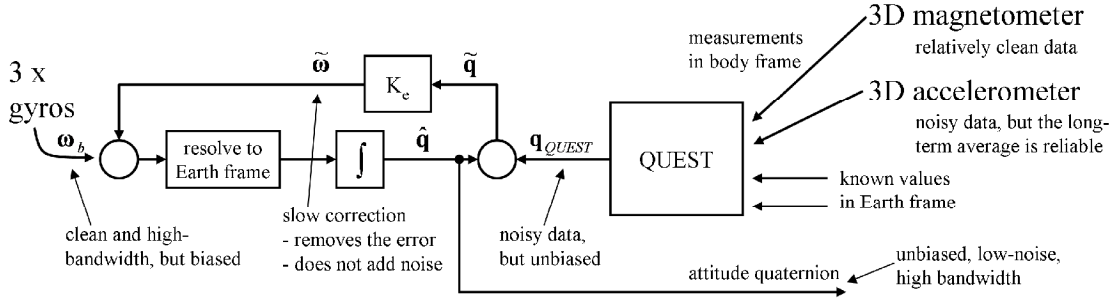


Figure 26. Attitude estimation

The error quaternion $\tilde{\mathbf{q}}$ is defined as the rotation needed to go from the estimated orientation quaternion $\hat{\mathbf{q}}$ to the quaternion obtained from the QUEST algorithm $\mathbf{q}_{QUEST}$:

$$\tilde{\mathbf{q}} = \mathbf{q}_{QUEST} \otimes \hat{\mathbf{q}}^* . \tag{112}$$

There are two values possible for $\tilde{\mathbf{q}}$. Only one will do the rotation with a minimum angle. It corresponds to the case when the first component of the quaternion, $\tilde{q}_0 = \cos(\alpha/2)$ is positive. If $\tilde{\mathbf{q}}$ is obtained with $\tilde{q}_0 < 0$ then $-\tilde{\mathbf{q}}$ will be used instead.

The error quaternion corrects the estimate by generating an extra angular velocity $\tilde{\boldsymbol{\omega}}$ in the body frame on top of the gyroscope measurements. In the next equation for the dynamics of the estimate of the attitude quaternion, $\tilde{\boldsymbol{\omega}}$ is augmented to a quaternion $\overline{\tilde{\omega}}$.

$$\dot{\hat{\mathbf{q}}} = \frac{1}{2}\hat{\mathbf{q}} \otimes \left(\overline{\boldsymbol{\omega}}_b + \overline{\tilde{\omega}} \right) \tag{113}$$

To find $\tilde{\boldsymbol{\omega}}$, $\tilde{\mathbf{q}}$ is written in the geometric form and its rotation axis is resolved to body coordinates:

$$\tilde{\mathbf{q}} = \begin{bmatrix} \cos(\alpha/2) \\ \sin(\alpha/2) \cdot \mathbf{r}_e \end{bmatrix} \text{ and } \tilde{\mathbf{q}}_b = \begin{bmatrix} \cos(\alpha/2) \\ \sin(\alpha/2) \cdot \mathbf{r}_b \end{bmatrix}, \tag{114}$$

with

$$\mathbf{r}_b = \mathbf{q}^* \otimes \mathbf{r}_e \otimes \mathbf{q} . \tag{115}$$

It is reasonable to make a small-angle assumption about the rotation angle α in the error quaternion. In this case equation (114) becomes

$$\tilde{\mathbf{q}} \approx \begin{bmatrix} 1 \\ \alpha/2 \cdot \mathbf{r}_e \end{bmatrix} \text{ and } \tilde{\mathbf{q}}_b \approx \begin{bmatrix} 1 \\ \alpha/2 \cdot \mathbf{r}_b \end{bmatrix}. \tag{116}$$

A proportional law is chosen to get $\tilde{\boldsymbol{\omega}}$:

$$\tilde{\boldsymbol{\omega}} = K_e \alpha \cdot \mathbf{r}_b \,, \tag{117}$$

where K_e is the time constant for the convergence of the estimate to the measured value. It is chosen to be small enough for noise rejection from measurements but large enough to allow for a good cancelation of the gyroscopes drift. Equation (113) becomes:

$$\dot{\hat{\mathbf{q}}} = \frac{1}{2}\hat{\mathbf{q}} \otimes \bar{\boldsymbol{\omega}}_b + \frac{1}{2}\hat{\mathbf{q}} \otimes \left(K_e \alpha \cdot \hat{\mathbf{q}}^* \otimes \mathbf{r}_e \otimes \hat{\mathbf{q}} \right) \tag{118}$$

and finally

$$\dot{\hat{\mathbf{q}}} = \frac{1}{2}\hat{\mathbf{q}} \otimes \bar{\boldsymbol{\omega}}_b + K_e \begin{bmatrix} 0 \\ \tilde{q}_1 \\ \tilde{q}_2 \\ \tilde{q}_3 \end{bmatrix} \otimes \hat{\mathbf{q}} \,. \tag{119}$$

Equation (119) is integrated using a method that maintains the unit norm of the rotation quaternion [29]. The integration over a sample period T is given by:

$$\hat{\mathbf{q}}(t+T) = \hat{\mathbf{q}}(t) \otimes e^{\int_t^{t+T} \mathbf{q}^*(\tau) \otimes \dot{\mathbf{q}}(\tau) d\tau} \,. \tag{120}$$

In order to be able to do the above integration in Simulink, a special S-function block was created. It overrides the built-in integration algorithms and performs the quaternion integration separately.

4.5. Attitude Controller

Motor Gain Estimation

The identical behavior of the motors is very important to keep the quadrotor stable in hover. The gains a_i from equation (107) are estimated on-line by comparing the measurements of Ω_i with their expected value $\hat{\Omega}_i$ obtained by integrating (107). The dynamics of the estimated gain is based on the estimation error:

$$\dot{a}_i = K_g \left(\Omega_{i\ meas} - \hat{\Omega}_i \right). \tag{121}$$

To avoid adaptation in unwanted operating points, the gain K_g is forced to zero at rotor speeds far from the nominal speed for hover.

Yaw and Tilt Errors

The quaternion formulation makes it easy to represent the orientation of a solid body relative to a reference axes system in a natural way by defining an axis of rotation $\mathbf{r}$ and an angle α for the amount of rotation. The attitude controller receives a desired orientation $\mathbf{q}_d$ and has to generate the right moment commands to rotate the vehicle from the current orientation $\mathbf{q}$ to $\mathbf{q}_d$. The error quaternion $\tilde{\mathbf{q}}$ is defined below:

$$\mathbf{q}_d = \tilde{\mathbf{q}} \otimes \mathbf{q} \tag{122}$$

$$\tilde{\mathbf{q}} = \mathbf{q}_d \otimes \mathbf{q}^* \tag{123}$$

The rotation axis of the error quaternion is resolved to the inertial (reference) coordinate system. In order to see what rotations are necessary in the body coordinates, the error quaternion will also be expressed in body coordinates as $\tilde{\mathbf{q}}_b$:

$$\tilde{\mathbf{q}} = \begin{bmatrix} \cos(\alpha/2) \\ \sin(\alpha/2) \cdot \mathbf{r}_e \end{bmatrix} \tag{124}$$

$$\mathbf{r}_b = \mathbf{q}^* \otimes \mathbf{r}_e \otimes \mathbf{q} \tag{125}$$

$$\tilde{\mathbf{q}}_b = \begin{bmatrix} \cos(\alpha/2) \\ \sin(\alpha/2) \cdot \mathbf{r}_b \end{bmatrix} \tag{126}$$

As opposed to the usual approach based on Euler angles (Figure 27) where yaw is an angle in the horizontal plane and tilt is expressed using the pitch and roll angles, the attitude controller presented in this chapter uses a different parameterization, relative to the body coordinate system.

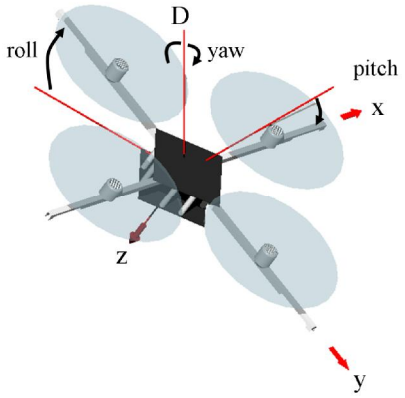


Figure 27. Attitude error representation using Euler angles

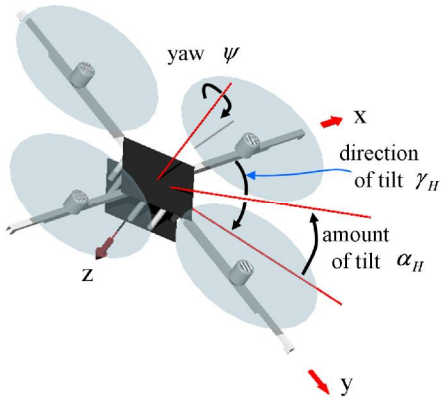


Figure 28. Attitude error representation in the body frame

The yaw angle ψ defines a rotation around the z axis and the tilt is defined by two angles with a different meaning: a tilt direction γ_H in the $x-y$ plane and the tilt amount α_H (Figure 28). This parameterization is now linear and independent of the orientation of the vehicle relative to the inertial frame. The yaw and the tilt errors are defined by $\mathbf{q}_V$ and $\mathbf{q}_H$:

$$\tilde{\mathbf{q}}_b = \mathbf{q}_H \otimes \mathbf{q}_V \tag{127}$$

or in the geometric representation:

$$\begin{bmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \end{bmatrix} = \begin{bmatrix} \cos(\alpha_H/2) \\ \sin(\alpha_H/2)\cdot r_x \\ \sin(\alpha_H/2)\cdot r_y \\ \sin(\alpha_H/2)\cdot 0 \end{bmatrix} \otimes \begin{bmatrix} \cos(\psi/2) \\ \sin(\psi/2)\cdot 0 \\ \sin(\psi/2)\cdot 0 \\ \sin(\psi/2)\cdot 1 \end{bmatrix}. \tag{128}$$

The amount of tilt is positive by convention:

$$\alpha_H = \arccos\left[1-2\left(q_1^2+q_2^2\right)\right], \quad 0 \le \alpha_H < \pi \tag{129}$$

The function $\arctan_2(a,b)$ considers the sign of a and b, and places the angle in the correct quadrant.

$$\psi = 2\arctan_2(q_3, q_0)\ , \quad -\pi \le \psi < \pi \tag{130}$$

The vehicle is tilted around the axis defined by r_x and r_y:

$$r_x = \frac{\cos(\psi/2)q_1 - \sin(\psi/2)q_2}{\sin(\alpha_H/2)}, \tag{131}$$

$$r_y = \frac{\sin(\psi/2)q_1 + \cos(\psi/2)q_2}{\sin(\alpha_H/2)}. \tag{132}$$

β_H is the direction of the tilt axis in the $x-y$ plane:

$$\beta_H = \arctan_2\left(r_y, r_x\right) \tag{133}$$

and γ_H is the direction of the tilt:

$$\gamma_H = \arctan_2\left(r_x, -r_y\right). \tag{134}$$

Attitude Stabilization

The attitude is stabilized by a yaw controller and by a tilt controller. There are no separate controllers for the x and for the y axes. The desired moments around each axis are

$$\begin{aligned} M_x &= M_{PH}\cos\beta_H + M_{Dx} \\ M_y &= M_{PH}\sin\beta_H + M_{Dy} \\ M_z &= M_{Pz} + M_{Dz} \end{aligned} \tag{135}$$

where

$$\begin{aligned} M_{PH} &= K_{PH}\alpha_H \\ M_{Pz} &= K_{Pz}\psi \end{aligned} \tag{136}$$

are the proportional controllers for tilt and yaw. The derivative action uses measurements directly from the gyroscopes:

$$\begin{aligned} M_{Dx} &= -K_{DH}\omega_x \\ M_{Dy} &= -K_{DH}\omega_y \\ M_{Dz} &= -K_{Dz}\omega_z \end{aligned} \tag{137}$$

The forces that go as commands to the motors are obtained by inverting equation (108) and the final PWM commands by using equations (110) and (111).

4.6. Altitude Controller

An ultrasound range sensor is used to measure altitude. The force necessary to cancel the weight of the quadrotor is generated by a feed-forward path. The vertical component of the F_z force (along the D axis) is commanded to be equal to the weight plus some contribution from the PID controller for the altitude:

$$F_z = \frac{mg + F_{PIDz}}{K_{tilt}} \tag{138}$$

where

$$K_{tilt} = \begin{bmatrix} 0 & 0 & 1 \end{bmatrix} \cdot \left(q \otimes \begin{bmatrix} 0 \\ 0 \\ -1 \end{bmatrix} \otimes q^* \right) \tag{139}$$

4.7. Pilot Control

The first objective of the actual implementation of the control system for the quadrotor is to allow an untrained person to fly it using a remote control unit. Usually the pilot commands are resolved to the body frame of the vehicle, making control very

difficult (Figure 29). It is much easier for a pilot to control the quadrotor if the commands are resolved to the pilot's reference frame instead (Figure 30). The quadrotor is able to hover autonomously and to respond in an intuitive manner to the pilot commands. A second quadrotor electronic board was installed on the remote control (Figure 31). The pilot tilts the remote control in any direction and so generates a heading command. The amount of tilt is the velocity command.

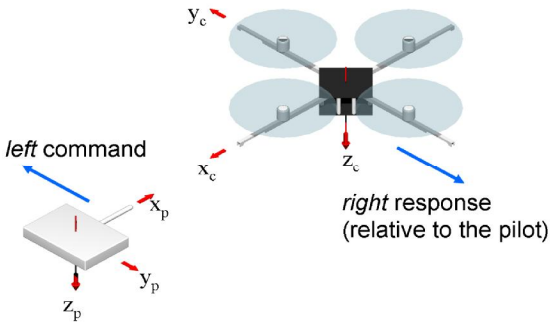


Figure 29. Pilot control in the vehicle body coordinates

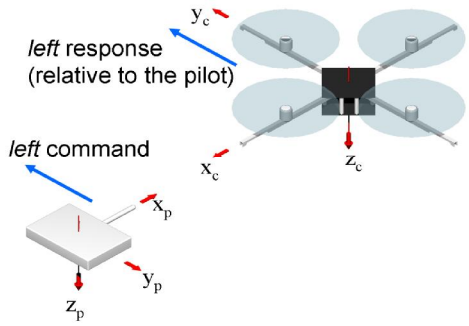


Figure 30. Pilot control in the pilot-referenced coordinates

Figure 31. A second quadrotor sensor board detects the pilot's orientation

Using the same development as in the case of the quadrotor error quaternion $\tilde{\mathbf{q}}$, the remote control has a tilt amount α_{rem} and a tilt direction γ_{rem} relative to the NED coordinate system. Using these angles, $\mathbf{q}_d$ for the quadrotor is generated in the following way:

$$\mathbf{q}_d = \begin{bmatrix} \cos(\theta/2) \\ \sin(\theta/2)\cdot \mathbf{r}_{de} \end{bmatrix} \tag{140}$$

where

$$\mathbf{r}_{de} = \begin{bmatrix} \sin\gamma_{rem} \\ -\cos\gamma_{rem} \\ 0 \end{bmatrix} \tag{141}$$

and

$$\theta = \arcsin\left(K_s \alpha_{rem}\right). \tag{142}$$

The gain K_s determines the sensitivity of the pilot commands.

Independent of the yaw angle and independent of the orientation of the pilot, using this definition of the commanded quaternion ensures that the quadrotor is tilting in the same direction in which the pilot tilts the remote control.

4.8. Experimental Data and Conclusion

The overall control structure is presented in Figure 32.

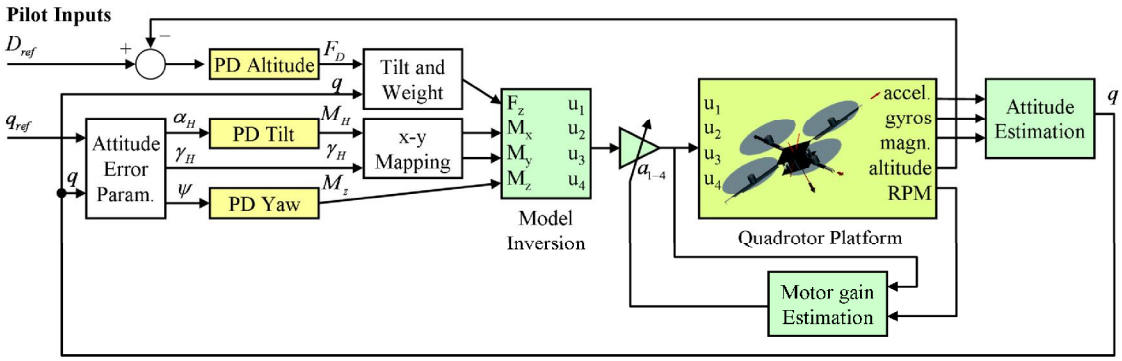


Figure 32. Overall control structure for the quadrotor

Experimental data for the attitude control with a horizontal reference is shown in Figure 33. The PD controller is able to keep the attitude close to horizontal within a tight margin. The small errors are not only a consequence of the controller performance, but also of that of the sensors.

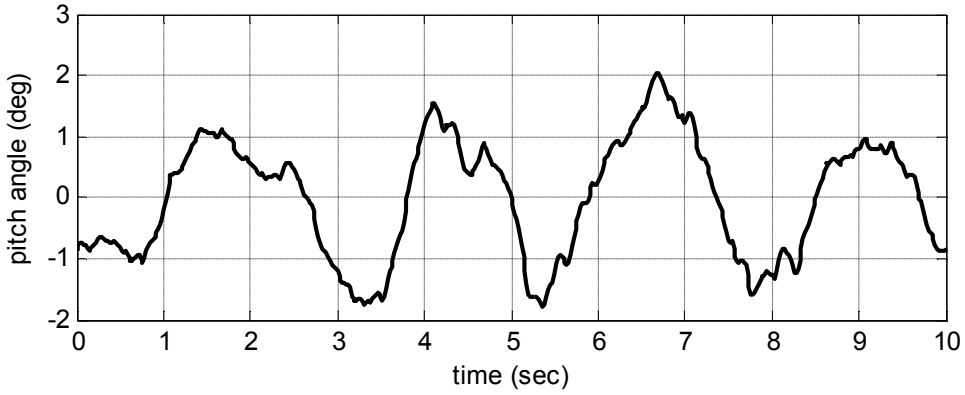


Figure 33. Experimental data recorded for the pitch angle during hover

This chapter presented the main components of a simple control system that was successful in stabilizing a custom-built quadrotor platform for hover. The most important issues were to estimate the attitude reliably and without significant noise using only the on-board sensors, to estimate the motor gains in order to ensure similar performance and to provide an intuitive control algorithm that can prove that the platform can be controlled satisfactory using simple PD controllers. The quaternion formulation allowed simple transformations from multiple coordinate systems and a natural representation of the controlled variables. The fact that the tilt was represented as a direction and the amount may prove useful for other types of controllers. An integral component could be added to compensate for the effects of the translational velocity. With this parameterization, the yaw motion would have little effect on the value of the integral term.

Figure 34. Experimental flight outside of the ARRI building

CHAPTER 5

APPROXIMATE DYNAMIC PROGRAMMING APPLIED TO UAV

5.1. Introduction

There is currently a dichotomy between optimal control and adaptive control. Adaptive Control algorithms learn online and give controllers with guaranteed performance for unknown systems. On the other hand, optimal control design is performed off line and requires full knowledge of the system dynamics. In this research we designed Optimal Adaptive Controllers, which learn online in real-time and converge to optimal control solutions. For linear time-invariant systems, these controllers solve the Riccati equation online in real-time by using data measured along the system trajectories. These results show how to approximately solve the optimal control problem for nonlinear systems online in real-time, while simultaneously guaranteeing that the closed-loop system is stable, i.e. that the state remains bounded. This solution requires knowledge of the plant dynamics, but in future work it is possible to implement algorithms that only know the structure of the system and not the exact dynamics.

The main focus of this chapter is to present different mechanisms for efficient learning by using as much information about the system and the environment as

possible. Learning speed is crucial for a real-time, real-life application that has to accomplish a useful task. The control algorithm isn't usually allowed to generate the best commands suitable for exploration and for learning, because this would defeat the purpose of having the controller in the first place, which is to follow a designated trajectory. The information gathered along the trajectory has to be used efficiently to improve the control policy. There is a big amount of data that has to be stored for such a task. The system is complex and has a large number of continuous state variables. The value function and the policy that corresponds to the infinite number of combinations of state variable values and possible commands have to be stored using a finite number of parameters. The coding of these two functions is made using function approximation with a modified version of radial basis function neurons. Due to their local effect on the approximation, the RBF neurons are best suited to hold information that corresponds to training data generated only around the current operating point, which is what one can obtain by following a normal trajectory without exploration. The usual approach of using multilayer perceptrons that have a global effect suffers from having to do a compromise between learning speed and the dispersion of the training samples. For samples that are concentrated around the operating point, learning has to be very slow to avoid deteriorating the approximation precision for states that are far away.

Two very important characteristics of learning are generalization and classification. The amount of information gathered by the system corresponds only to particular state trajectories and particular commands. Still, the value of being in a certain state and of using a certain command has to be estimated over an infinite

continuous space. The RBF neurons are able to interpolate between the specific points where data samples are stored. They don't provide a global solution, but they certainly cover the space around the states likely to be visited in normal conditions. The neural network structure is adaptive. Neurons are added or removed as needed. If for a specific operating point the existing neurons can't provide enough accuracy to store a new sample, then a new neuron is added in that point. The modified RBF neurons are created initially with a global effect in all dimensions. It is only on the dimensions where there is a need to discern between different values of the state variable that the effect is local. This mechanism allows neurons to partition the state space very efficiently. If some state variables do not affect the value function or the control policy corresponding to a certain region of the state space, then the neurons in the vicinity of that region are global on those dimensions. This organization of the RBF network falls in line with the idea that if the function to be approximated is not very complicated, then a reasonably small number of parameters should be sufficient to achieve a small error even if the number of dimensions of the input space is large. This applies to smooth and nice behaving functions. In the worst case, the number of parameters needed grows exponentially with the number of inputs. For the current implementation, the total number of neurons is kept at a reasonable value by pruning the ones in regions that have been visited in the distant past and thus diluting the approximation precision in those regions.

5.2. Background

Neural networks were used to design an adaptive controller in [30] and to learn the complete dynamics of the quadrotor online and to stabilize the platform using output feedback in [31].

All these projects and many others (as shown in section 4.2) use various techniques to stabilize the quadrotor, but they can't reach optimal performance. On-line ADP has a great potential in this direction, while also maintaining the adaptability and robustness of other algorithms. Still, the scale of the problem seems too big for standard ADP algorithms that are usually applied to systems with two or three states. Even more, we try to solve a trajectory tracking problem, not the regulation problem. A compromise has to be made. Instead of having overall optimality, the control algorithm is split around three smaller loops with similar behavior: translation, attitude and motor/propeller control. A global critic is maintained, but the overall behavior only converges to some correlated local optimums.

Physical analysis of dynamical systems using Lagrangian mechanics, Hamiltonian mechanics, etc. produces system descriptions in terms of nonlinear ordinary differential equations. Particularly prevalent are nonlinear ODEs in the state-space form

$$\dot{x} = f(x,u) \tag{143}$$

with the state $x(t) \in \boldsymbol{R}^n$ and control input $u(t) \in \boldsymbol{R}^m$ residing in continuous spaces. Many systems in aerospace, the automotive industry, process industry, robotics, and

elsewhere are conveniently put into this form. In addition to being continuous-state space and continuous-input space systems, in contrast to Markov decision processes (MDP) which have discrete states and actions, these dynamics are also continuous-time (CT) systems. For nonlinear systems, the policy iteration (PI) algorithm was first developed by Leake and Liu [32]. Three decades later it was introduced by Beard, Saridis, and Wen [33] as a feasible adaptive solution to the CT optimal control problem.

The bulk of research in ADP has been conducted for systems that operate in discrete-time (DT). Although the quadrotor model is a continuous-time model, all other signals outside the robot are sampled in discrete time. Therefore, we develop DT ADP control algorithms. Some clarification is required regarding the indices used to show the current time step and the signals available at each time step. Some of the existing literature does not necessarily consider what happens on a real sampled system and ignores the fact that signal propagation through the system and the algorithm computation require a certain amount of time. For simulations this issue is not very important, but for real-time applications things have to be defined very clearly.

At time step k the following two events take place at the dynamic system $\dot{x} = f(x,u)$: the state x_k is sampled, and the command u_k is presented to the system. There is a zero-order hold for the command, so u_k remains constant for the whole time interval $[k, k+1)$. Because u_k must be available concurrently with the sampling of x_k, there is no time left for the control algorithm to calculate u_k as a function of x_k. For a

practical sampled system subject to communication and data processing delays there is a delay of at least one sample period between the measurements and the commands.

There are standard methods for sampling or discretizing nonlinear continuous-time state space ODEs to obtain sampled data forms that are convenient for computer-based control [34]. The resulting systems unfold in discrete time and are generally of the state-space form

$$x_{k+1} = F(x_k, u_k) \tag{144}$$

with k the discrete time index. These systems satisfy the 1-step Markov property since their state at time $k+1$ only depends on the state and inputs at the previous time k. For good precision in estimating x_{k+1}, a Runge-Kutta algorithm can be used to find a solution to the ordinary differential equation $\dot{x} = f(x,u)$ with initial condition x_k and constant input u_k for the time interval $T = t_{k+1} - t_k$. If precision is not needed, the first-order approximation may be preferred:

$$x_{k+1} = x_k + Tf(x_k, u_k). \tag{145}$$

5.3. Reference Model

As shown in section 3.3 and in Figure 20, the quadrotor model can be split into three cascaded subsystems: translation, attitude and motors/propellers. For each subsystem, a reference model is introduced. The reference model generates reference accelerations ($\dot{\mathbf{\Omega}}_{ref}$, $\dot{\boldsymbol{\omega}}_{b\,ref}$, $\dot{\mathbf{V}}_{b\,ref}$) needed for tracking. A NN-based actor augments this signal and compensates for imprecision in the inverses of the f_ω, f_V and f_Ω functions.

The reference models are similar for the three loops. The one for the translation subsystem is given by:

$$\begin{aligned}
\mathbf{a}_{c\ ref} &= K_P\left(\mathbf{P}_c - \mathbf{P}_{ref}\right) + K_D\left(\mathbf{V}_{e\ c} - \mathbf{V}_{e\ ref}\right) \\
\dot{\mathbf{V}}_{e\ ref} &= \mathbf{a}_{cref} - \mathbf{a}_{aw} \\
\dot{\mathbf{P}}_{e\ ref} &= \mathbf{V}_{e\ ref}
\end{aligned} \tag{146}$$

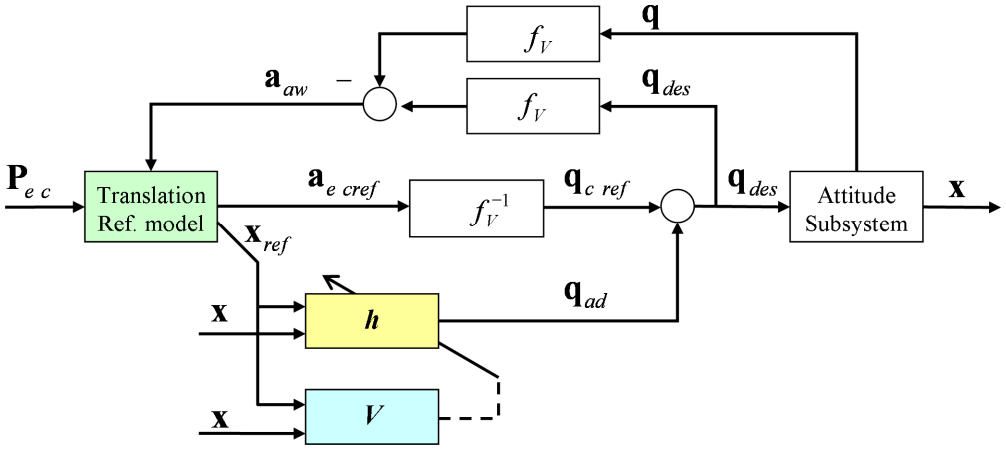


Figure 35. Control structure for the translation subsystem

The purpose of the $\mathbf{a}_{aw}$ variable is to prevent the reference model from demanding tracking if the inner loop actuators are saturated or if the inner loop dynamics is too slow. The actor does not see $\mathbf{x}_{ref}$ values that have no effect on the inner loop because of saturation or slow dynamics. The learning process can thus continue even during saturation. The gains in the reference model can be tuned manually in order to provide certain handling qualities to the vehicle, or can be obtained using an optimal algorithm. Saturation for the velocities can be introduced in the following manner:

$$\mathbf{a}_{c\ ref} = K_D\left[\max\left(\min\left(\frac{K_P}{K_D}\left(\mathbf{P}_c - \mathbf{P}_{ref}\right), \mathbf{V}_{\max}\right), \mathbf{V}_{\min}\right) + \left(\mathbf{V}_{e\ c} - \mathbf{V}_{e\ ref}\right)\right] \tag{147}$$

5.4. Approximate Dynamic Programming

An actor-critic structure is used for implementing efficient reinforcement learning (Figure 36).

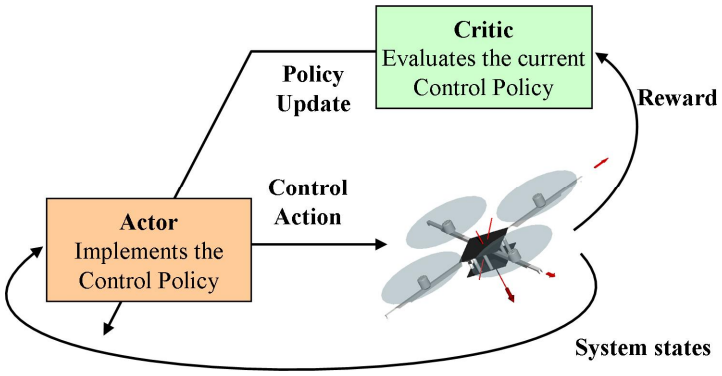


Figure 36. Generic ADP structure

Most ADP algorithms solve regulation problems. The quadrotor is designed to follow a certain trajectory and not just to hover. The ADP algorithm has to be modified to include an additional reference signal $\eta_k \in \boldsymbol{R}^n$ that is provided from outside the model. The tracking error is defined as

$$z_k = x_k - \eta_k \,. \tag{148}$$

The actor, or the control policy, is

$$u_k = h(x_{k-1}, z_{k-1}) \,. \tag{149}$$

It is important to observe that the command u can only be calculated as a function of previous states. This is a consequence of the fact that the sampling of the state variables and of the command is synchronous. Also note that u_k in this section represents a vector that contains the following signals:

$$u = \begin{bmatrix} \mathbf{q}_{ad} \\ a_{z\,ad} \\ \boldsymbol{\Omega}_{ad} \\ \mathbf{V}_{mot\,ad} \end{bmatrix} \tag{150}$$

The quadrotor has 17 states and only 4 control inputs ($\mathbf{u} = \mathbf{V}_{mot}$), thus it is very under-actuated. Three control loops with dynamic inversion are used to generate the 4 control signals. Some learning is forced to reside locally by splitting the actor into local actors for each of the three loops. Each actor works with a reduced set of states. The critic is global:

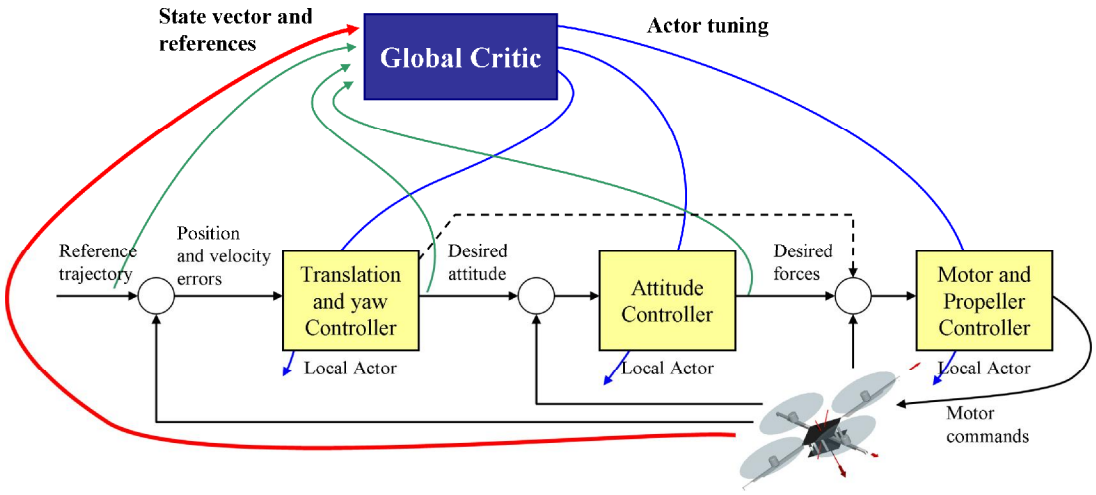


Figure 37. Distributed ADP structure for the quadrotor

The weights of the NN-based actors are set to generate known stable PD controllers at the initialization phase of the algorithm. They will depart from this configuration while training takes place.

The control policy is distributed in the three subsystems of the quadrotor: translation, attitude and motor/propeller control (Figure 37):

$$h(x_k, z_k) = \left[h^{1T}(x_k^1, z_k^1), h^{2T}(x_k^2, z_k^2), h^{3T}(x_k^3, z_k^3) \right]^T \tag{151}$$

where the variables x^i, z^i, u^i are subsets of the state vector, the tracking error vector and the command vector that are used for the control policies in each of the three loops.

The control structure from Figure 37 is inspired from the natural structures shown in Figures 5–6 and the subsumption architecture presented in Figure 7 in Chapter 2. Each local actor reacts very fast to stimuli in the same way a reflex does in the nervous system. Each of the three loops sits at a different hierarchical level. The loops can work independently, but their behavior is changed by inputs coming from the higher level as in the subsumption architecture. The timing scale is also different. At the lowest level is the motor/propeller subsystem, with the fastest dynamics. The attitude subsystem has slower dynamics, and the translation is the slowest. The highest hierarchical level is the critic, which can take decisions about new control policies at a completely different time scale. The trajectory reference can be generated using fuzzy-logic rules extracted from an experienced pilot or by using the formation control algorithm from Chapter 6.

The key idea of reinforcement learning generally and of dynamic programming in particular is the use of value functions to organize and structure the search of good policies. The notion of goal-directed optimal behavior is captured by defining a performance measure or value function

$$V_h(x_k, z_k) = \sum_{j=k}^{\infty} \gamma^{j-k} r\left(x_j, z_j, u_j\right) \tag{152}$$

with $0 < \gamma \leq 1$ a discount factor and $u_k = h(x_{k-1}, z_{k-1})$ a prescribed feedback control policy. This is known as the cost-to-go associated to the policy h and is a sum of discounted future costs from the current time k into the infinite horizon future if the policy h is used at every step. The discount factor reflects the fact that we are less concerned about costs acquired further into the future.

Function $r(x_k, z_k, u_k)$ is known as the utility, and is a measure of the one-step cost of control. For the tracking problem it is defined in the following manner [35–37]:

$$r\left(x_k, z_k, u_k\right) = z_k^T Q z_k + \left(w_{k+1} - w_k\right)^T R\left(w_{k+1} - w_k\right) + \left(u_k - u_{ek}\right)^T S\left(u_k - u_{ek}\right) \quad (153)$$

where Q, R, S are positive definite matrices, w_k is the deviation from the expected control

$$w_k = u_k - u_{ek} \quad (154)$$

and the expected control u_{ek} is the command given in optimal conditions with perfect tracking when the nominal subsystem in equation (145) is considered:

$$\eta_{k+1} = \eta_k + Tf(\eta_k, u_{ek}) \quad (155)$$

$$u_{ek} = f^{-1}\left(\eta_k, \frac{1}{T}\left(\eta_{k+1} - \eta_k\right)\right). \quad (156)$$

The objective of optimal control is to select the policy that minimizes the cost to obtain

$$V^*(x_k, z_k) = \min_h \left(\sum_{j=k}^{\infty} \gamma^{j-k} r\left(x_j, z_j, h\left(x_{j-1}, z_{j-1}\right)\right)\right) \quad (157)$$

which is known as the optimal cost, or optimal value. Then, the optimal control policies are given by

$$h^*(x_k, z_k) = \arg\min_h \left(\sum_{j=k}^{\infty} \gamma^{j-k} r\left(x_j, z_j, h\left(x_{j-1}, z_{j-1}\right)\right) \right). \tag{158}$$

By writing (152) as

$$V_h(x_k, z_k) = r\left(x_k, z_k, h\left(x_{k-1}, z_{k-1}\right)\right) + \gamma V_h(x_{k+1}, z_{k+1}), \quad V_h(0) = 0 \tag{159}$$

instead of evaluating the infinite sum (152), one can solve the difference equation to obtain the value of using the current policy $u_k = h(x_{k-1}, z_{k-1})$. This is a nonlinear Lyapunov equation known as the Bellman equation. Evaluating the value of a current policy using the Bellman equation is the first key concept in developing reinforcement learning techniques. Bellman's optimality principle [38] states that "an optimal policy has the property that no matter what the previous decisions have been, the remaining decisions must constitute an optimal policy with regard to the state resulting from those previous decisions". This gives the Bellman optimality equation or the discrete-time Hamilton-Jacobi-Bellman (HJB) equation:

$$V^*(x_k, z_k) = \min_h \left(r\left(x_k, z_k, h\left(x_{k-1}, z_{k-1}\right)\right) + \gamma V^*(x_{k+1}, z_{k+1}) \right). \tag{160}$$

Determining optimal controllers using these equations is considerably easier since the optimum value is inside the minimization argument.

Since the optimal policy must be known at time $k+1$ to use (160) to determine the optimal policy at time k, Bellman's principle yields a backwards-in-time procedure for solving the optimal control problem. This is by nature an off-line planning method and full knowledge of the system dynamics $f(x_k, u_k)$ is needed. We prefer to avoid

using the system dynamics whenever possible to allow for adaptation to changing dynamics, and we also need on-line methods.

ADP will be used to do on-line reinforcement learning in real-time for solving the optimal control problem by using data measured along system trajectories [39]. Q learning, introduced by Watkins [40, 41] provides an alternative path to take partial derivatives with respect to the control input that does not go through the system, allowing optimization without the knowledge of the system dynamics. The quality function Q associated with the policy h is defined as

$$Q_h\left(x_k, z_k, u_k\right) = r\left(x_k, z_k, u_k\right) + \gamma V_h\left(x_{k+1}, z_{k+1}\right). \tag{161}$$

The policy $u_k = h\left(x_{k-1}, z_{k-1}\right)$ has to be admissible, meaning that it must be stabilizing and it must yield a finite cost $V_h\left(x_k, z_k\right)$. That is why the RBF neural networks that approximate the policies h^1, h^2, h^3 are initialized at the beginning with the necessary structure and the weights corresponding to known stabilizing PD controllers. The optimal Q function is defined as

$$Q^*\left(x_k, z_k, u_k\right) = r\left(x_k, z_k, u_k\right) + \gamma V^*\left(x_{k+1}, z_{k+1}\right). \tag{162}$$

In terms of Q^* the Bellman optimality equation can be written in the very simple form

$$V^*\left(x_k, z_k\right) = \min_u\left(Q^*\left(x_k, z_k, u\right)\right) \tag{163}$$

and the optimal control as

$$h^*\left(x_k, z_k\right) = \arg\min_u\left(Q^*\left(x_k, z_k, u\right)\right). \tag{164}$$

From the two equations above one can write the value function as

$$V^*\left(x_k, z_k\right) = Q^*\left(x_k, z_k, h^*\left(x_k, z_k\right)\right) \tag{165}$$

which can be used to determine the fixed-point equation for Q^*:

$$Q^*\left(x_k, z_k, u_k\right) = r\left(x_k, z_k, u_k\right) + \gamma Q^*\left(x_{k+1}, z_{k+1}, h^*(x_{k+1}, z_{k+1})\right). \tag{166}$$

In the absence of control constraints, the control policy $u = h\left(x_k, z_k\right)$ that generates the minimum of the value function is obtained by solving numerically

$$\frac{\partial}{\partial u} Q^*\left(x_k, z_k, u\right) = 0. \tag{167}$$

A RBF neural network is used to approximate the Q function. It has the general form

$$Q_h(x, z, u) = W^T \phi(x, z, u). \tag{168}$$

Temporal-difference learning uses experience to solve the prediction problem. Prediction error is introduced in terms of the Bellman equation as

$$e_{k-1} = r\left(x_{k-1}, z_{k-1}, h\left(x_{k-2}, z_{k-2}\right)\right) + \gamma W^T \phi\left(x_k, z_k, u_k\right) - Q\left(x_{k-1}, z_{k-1}, u_{k-1}\right) \tag{169}$$

If the Bellman equation holds, $e_{k-1} = 0$ and the equation above can be solved for the Q function:

$$Q\left(x_{k-1}, z_{k-1}, u_{k-1}\right) = r\left(x_{k-1}, z_{k-1}, h\left(x_{k-2}, z_{k-2}\right)\right) + \gamma W^T \phi\left(x_k, z_k, u_k\right). \tag{170}$$

Once the value of the Q function at $(x_{k-1}, z_{k-1}, u_{k-1})$ is known, a backup of it is made into the RBF neural network by adjusting the weights W and/or by adding more neurons and by reconfiguring their other parameters. This is a separate process that just needs to know the $\left(x, z, u\right)$ coordinates and the new value to store. The method is available in [42].

$$Q(x_{k-1}, z_{k-1}, u_{k-1}) = W^T \phi(x_{k-1}, z_{k-1}, u_{k-1}) + \\ \alpha\left[r(x_{k-1}, z_{k-1}, h(x_{k-2}, z_{k-2})) + \gamma W^T \phi(x_k, z_k, u_k) - W^T \phi(x_{k-1}, z_{k-1}, u_{k-1}) \right] \tag{171}$$

which means $Q_{stored} = Q_{old} + \alpha(Q_{new} - Q_{old})$ with $0 < \alpha < 1$. As it can be seen, the update of the Q value is not made completely towards the new value. This slows down the learning, but adds robustness.

The policy update step is done by solving

$$\frac{\partial}{\partial u} Q(x_k, z_k, u) = 0 \tag{172}$$

after the new Q value was stored using a gradient-descent numerical search algorithm. The value for $h(x_k, z_k) = u$ is stored into the actor RBF neural network using the same mechanism as before:

$$h(x_k, z_k) = U^T \sigma(x_k, z_k) + \beta\left[u - U^T \sigma(x_k, z_k) \right]. \tag{173}$$

Faster learning and improved robustness can be obtained by extending the learning process back in time and also into the future. Based on the new values for the Q function, training can be done again for a list of previously visited states. The algorithm maintains a fixed-size buffer of past values for (x, z, u). Equations (171)-(173) are applied in order for the time indices $k-2, k-3, ..., k-d-1$ where d is the depth of the buffer. When this mechanism is used, the α and β constants must have lower values. Simulation into the future is made using the system model in equation (145) to obtain the future states likely to be reached. If the model is precise, a longer horizon can be chosen. Equations (171)-(173) are applied this time for the time indices

$k, k+1, k+2, \ldots$. This mechanism allows the pre-training of the ADP neural networks for states that have not been visited before. The system model might not be very precise, but it can still provide better information than a completely untrained neural network. To prevent the model from affecting data that was already learned well, only the neurons with an old update timestamp are trained.

5.5. Global vs. Local Learning

At any moment in time, for the quadrotor in normal operating conditions only training data from near the operating point is available. There is no way of doing NN training based on samples from the entire state space because a very long time has to be spent collecting them. NN with global activation functions can only be trained reliably using batch methods with samples that cover the entire input domain. If training is attempted using samples concentrated into a specific region, the NN will adapt to that region and will *forget* what it has learned for the other regions of the input domain. That is why the learning algorithms for these networks either:

- Are very slow in order to do only small changes along the state-space trajectory, or
- Require very big amplitudes for the probing noise to guarantee that every region of the state space is visited often.

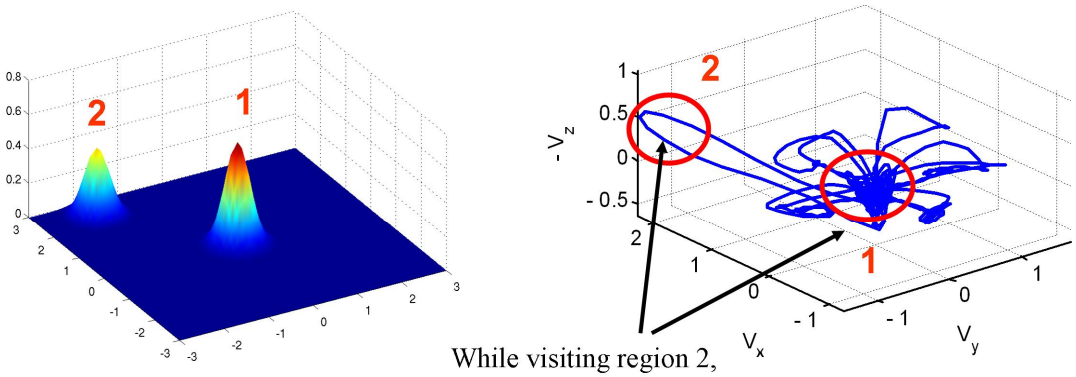


Figure 38. Local activation functions

Sampling the entire state-space for learning can be computationally expensive and impossible to realize with a real system due to time and energy constraints and also because the system has to do something useful like following a designated trajectory. In the short term, sampling along the system trajectory generated by the policy focuses learning to the states that are actually commonly occurring. In the long term, learning can suffer because the value function for these states is already very close to the optimal value and further learning does not help, and on the other hand there is not enough information for less common states. In Figure 38 the state space is not fully explored and there are long periods of time when the velocity only resides in a limited zone of the state space. This behavior does not follow the two requirements above and neural networks with global activation functions can't be trained with this data.

One important requirement for efficient learning is the following: optimization for the current operating point should not lose information stored in the past for the other operating points. When the system reaches a previously visited operating point, it

should not have to re-learn everything again. RBF neural networks allow local training with data taken from a restricted region of the state space. Training can be done at every sample or in batches. Every time only a few neurons that have a significant output for the current operating point are affected by learning. Data learned in the past for other operating points is preserved and immediately available when needed. This is very useful for cases when the operating point has a sudden jump due to severe disturbances. An algorithm that only has information around the current operating point would fail immediately when such a situation occurs.

5.6. The Curse of Dimensionality

The actor acts as a nonlinear function approximator. Normally we have

$$u_{k+1} = h(x_k) \tag{174}$$

In the quadrotor case, because the reference is not zero and the system is nonlinear, we need

$$u_{k+1} = h(x_k, z_k) \tag{175}$$

or for better for learning the context and error can be used:

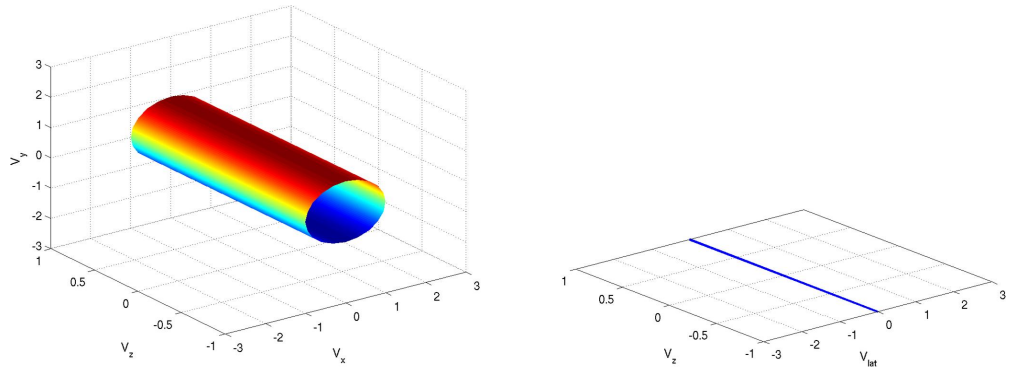
$$u_{k+1} = h(x_k, z_k - x_k) \tag{176}$$

For each of the position, attitude and motor/propeller loops the state vector includes the local states and the external states that have a big coupling effect on the loop performance. It is easy to see that this way the input space can easily have n=14 or more dimensions. A RBF neural network with the neurons placed on a grid with N elements in each dimension would require N^n neurons. For $N = 5$ and $n = 14$, $6 \cdot 10^9$

are required. Placing neurons on a grid is no better than a look-up table. The solutions to reducing the number of neurons are the following:

- preprocess the states to provide signals with physical significance as inputs,
- combine multiple states into a lower dimension signal, or
- map multiple equivalent regions from the state-space into only one

One simple example of removing a dimension is presented next. From the aerodynamic equations it can be seen that it is not the V_x and V_y velocities that directly affect the rotor forces, but their sum. If all the three velocities are inputs to the actor NN, the locus of a certain lateral velocity is a cylinder. Every time that specific velocity is reached, a different neuron close to the cylinder may be updated.



$V_x,\ V_y, V_z$ as inputs to the NN $\sqrt{V_x^2+V_y^2},\ V_z$ as inputs to the NN

Figure 39. Removing a dimension from the NN input

Learning can be made much faster by concentrating it from a cylinder to a line. This way a dimension can be removed from the input space of the NN.

Input processing for the neural networks can also speed up and concentrate learning. The usual approach of expressing the attitude by using Euler angles implies

that yaw error is an angle in the Earth horizontal plane (around the Down axis) and tilt error is expressed as pitch and roll error angles. The errors are resolved to Earth coordinates and this adds nonlinearities and couplings. Furthermore, the errors are dependent on the attitude of the vehicle relative to the Earth. This is very bad for learning because a certain amount of tilt error can have an infinite number of parameterizations.

In the current approach yaw error is an angle ψ in the *x*-*y* plane of the vehicle (around the *z* axis) and tilt is expressed as a tilt direction γ_H in the *x*-*y* plane and a tilt amount α_H. The attitude errors are resolved to body coordinates and this makes them independent of the attitude of the vehicle relative to the Earth. Learning can be much faster and focused because a certain amount of tilt error is always parameterized the same way. They also have better physical significance because they express directly what corrective movements the vehicle has to make in its own coordinate system, allowing for a direct mapping to the actuators.

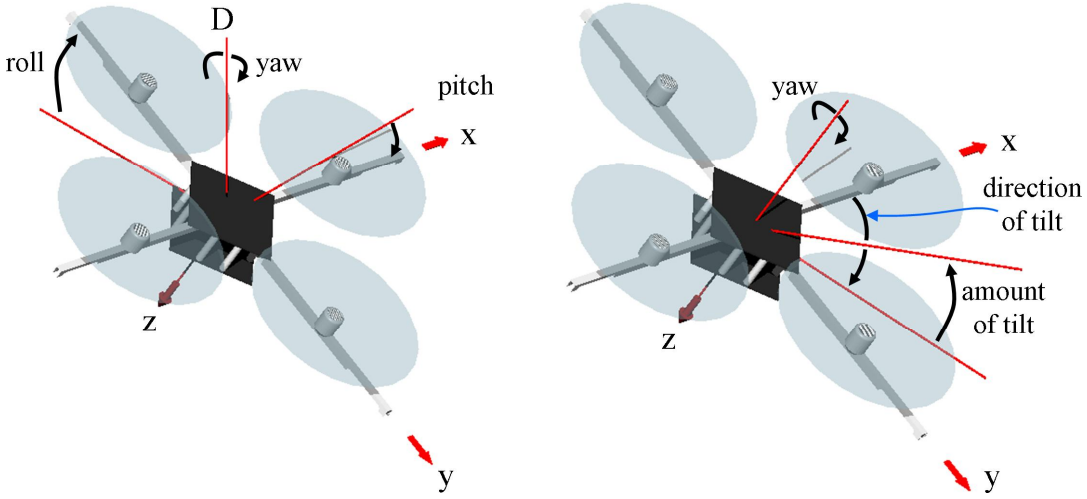


Figure 40. Standard and new error parameterization

5.7. Local Activation Basis Functions for Function Approximation

The actor neural network starts with a low number of wide RBF neurons that model a PD controller. The neurons are set to have infinite width on the dimensions where the input does not influence the output. This makes the quadrotor able to fly without any initial learning. Every time a training procedure is started, for any operating point, the algorithm tries to tune only the neurons that are active for that operating point. Each neuron has a linked list with its neighbors for fast real-time searching. If tuning the existing neurons can't offer enough precision, one more is added at the current operating point.

Each neuron has a time value associated with it from when it was last updated. The ones with the oldest values are eliminated if by taking them out and retraining the neighbors the precision doesn't go below a certain limit. This mechanism of adding neurons where needed and pruning the old ones without diluting the precision too much

allows good function approximation for recent data and reasonable approximation for old data that might be slightly outdated anyway.

5.8. Efficient Learning – Exploration vs. Exploitation

For a real-time system that can crash at the first mistake, learning has to be very efficient. For a practical system, there is a conflict between learning and doing a useful job (Figure 41).

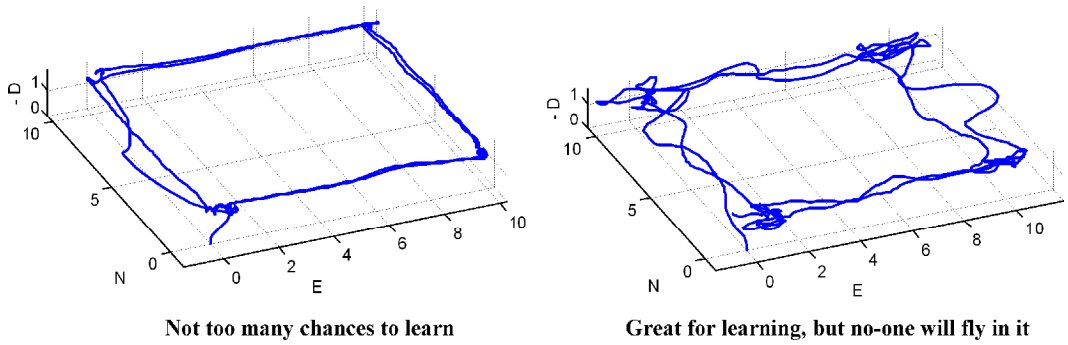


Figure 41. Exploitation versus exploration

An algorithm that requires persistence of excitation is not exciting for people in the industry. They work hard to make things smooth and use minimum energy. Under these conditions learning has to use whatever data is available during the normal operation of the vehicle. Having limited information for learning and limited freedom in deviating from the prescribed trajectory means that some information has to be obtained from another source. The best one is the actual system model. This seems to exclude from the start the model-free ADP algorithms. Other reasons are that they need to explore the whole state space numerous times. This can be done with simple systems that have 2 or at most 3 states. For an under-actuated system with 17 states the problem

becomes intractable. It is true that only 14 that have to be explored because the North, East and sometimes the Down positions do not affect the dynamics, but it is still a large state space.

Extensive exploration is time-consuming. Large deviations from the prescribed operating point are not desired or allowed. Unnecessary maneuvers increase energy consumption and chances of failure. Learning is also usually too slow for practical applications. There is another question: how does one place an under-actuated system that he doesn't know yet how to control in different representative points of the state space?

Model-based ADP algorithms can be made to only need local exploration around the nominal operating point. The exploration space is reduced by a few orders of magnitude. The amplitude of the probing noise can be smaller. Learning can be guided and it becomes much faster and more focused. Reasonable modeling errors can be also accommodated.

5.9. Simulation Results

The control algorithms are implemented in Simulink and run directly in the Matlab environment in normal or accelerated mode (Figure 42), allowing an instant transition between design and experiments. The same model can run a simulation or can control a real quadrotor. A special S-function block was created to allow the Simulink model to receive sensor data from the quadrotor and to send back the motor commands. The block communicates via USB with a base-station module connected to the

computer. Simulink generates and compiles C code for the model and is able to run it in real-time.

The simulation is done in the presence of either disturbing wind, weight imbalance or both at the same time. The quadrotor flies in a loop in the shape of a 10 x 10 meters square two times for 160 seconds. The wind has 5 m/s in the North direction and 5 m/s in the East direction. The ideal trajectory is a square formed by the points (0,0)-(0,10)-(10,10)-(10,0) meters. In Figures 43 and 44 the trajectory is shown first without adaptation and then with adaptation. Only two or one dimension are selected for plotting for the NN weights. When both wind and weight imbalance are present, the quadrotor becomes unstable without adaptation. In Figure 43 it is easy to observe the consequences of poor exploration of the state space. The adaptation only affects a few neurons corresponding to the states and the references that have been visited or applied.

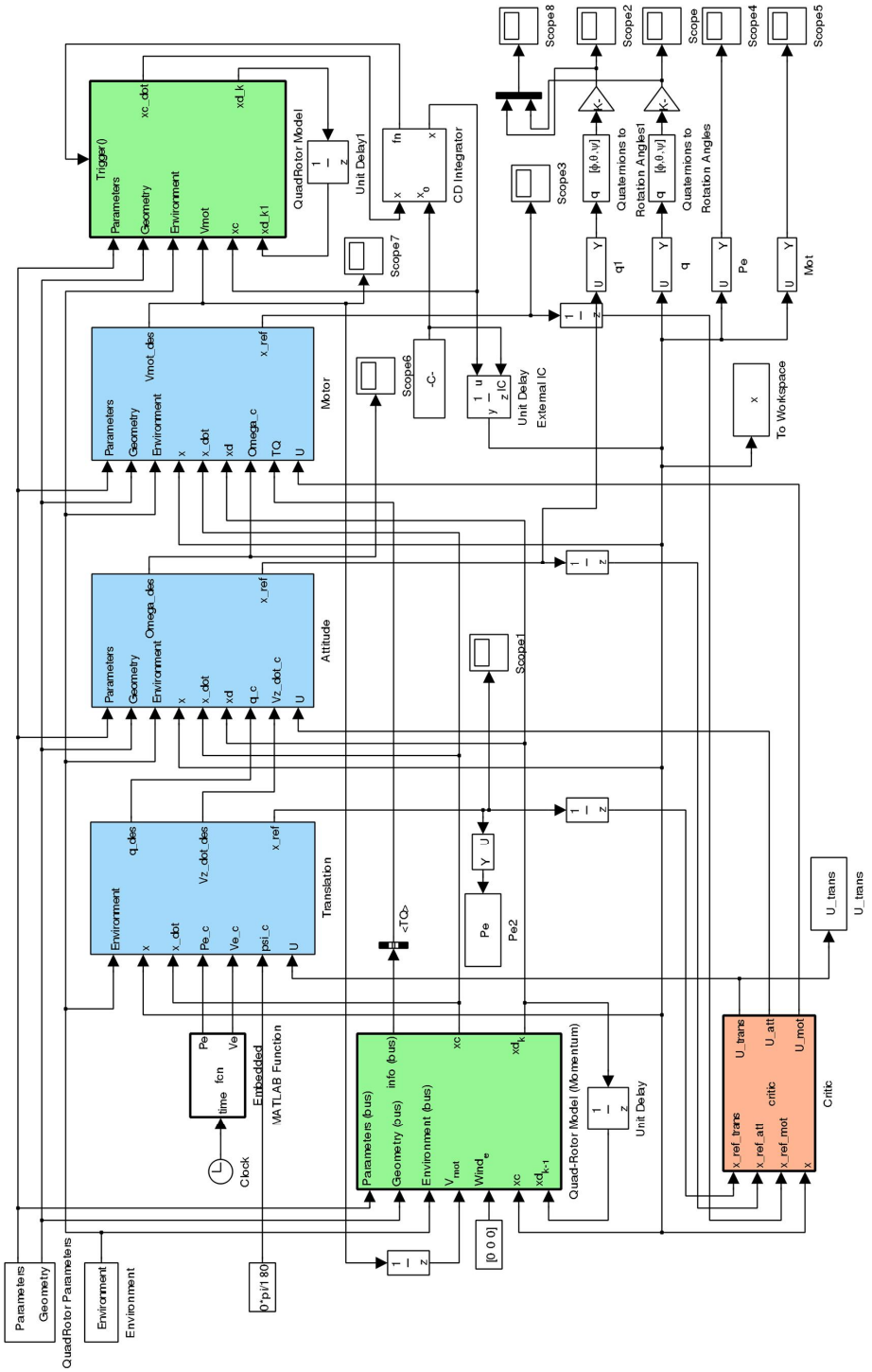


Figure 42. Simulink implementation for simulation and experiments

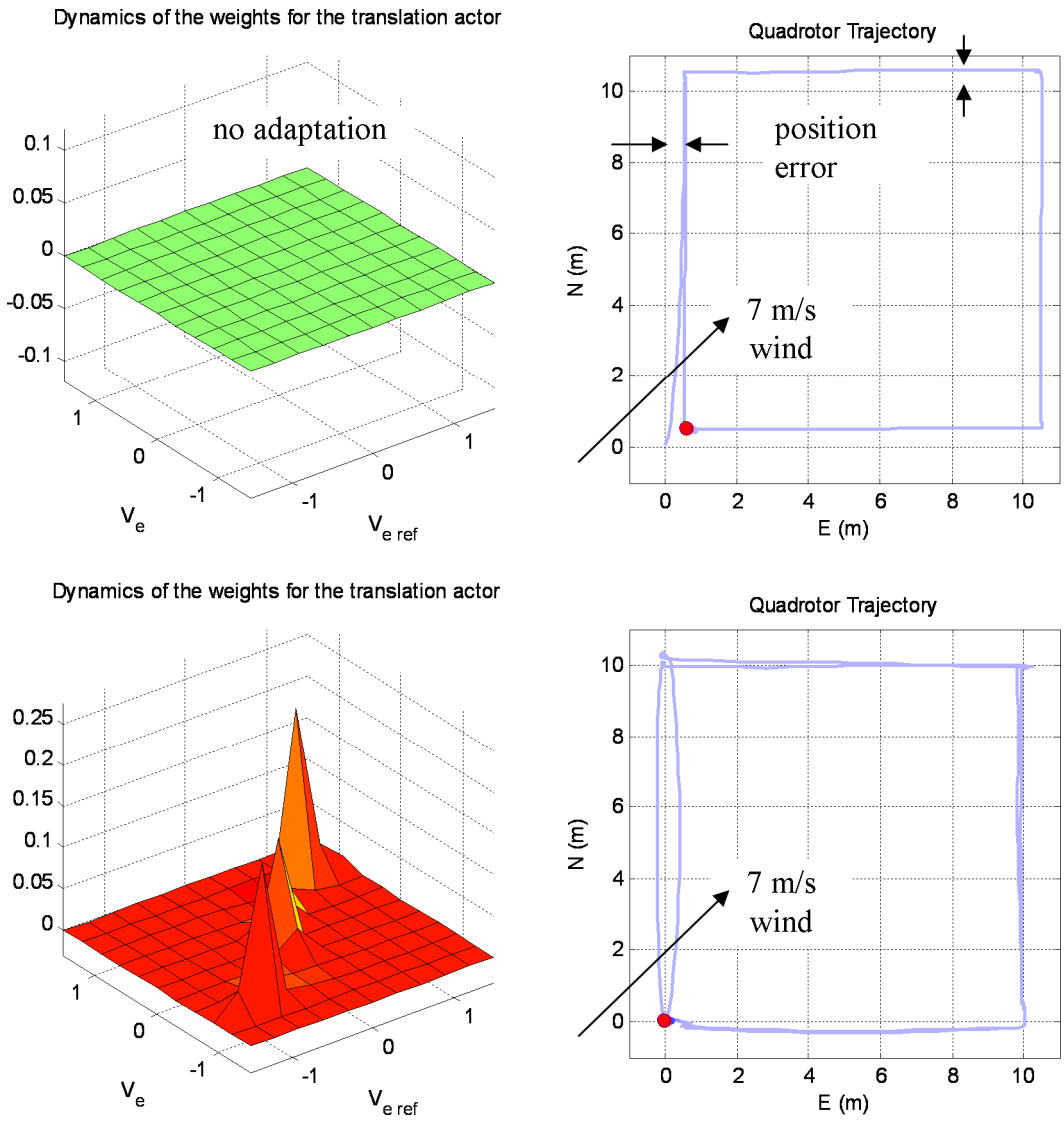


Figure 43. Wind disturbance

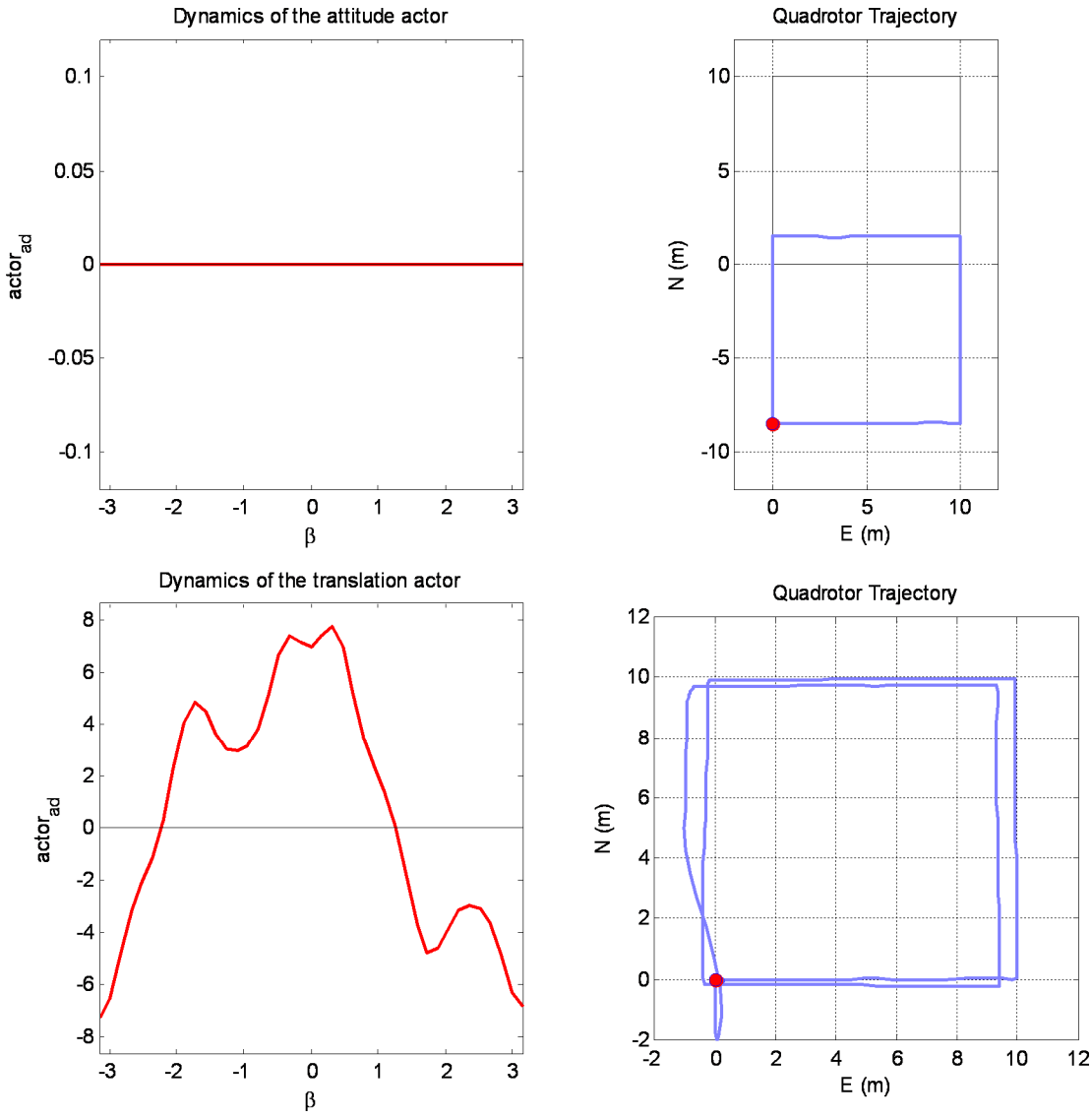


Figure 44. Weight imbalance

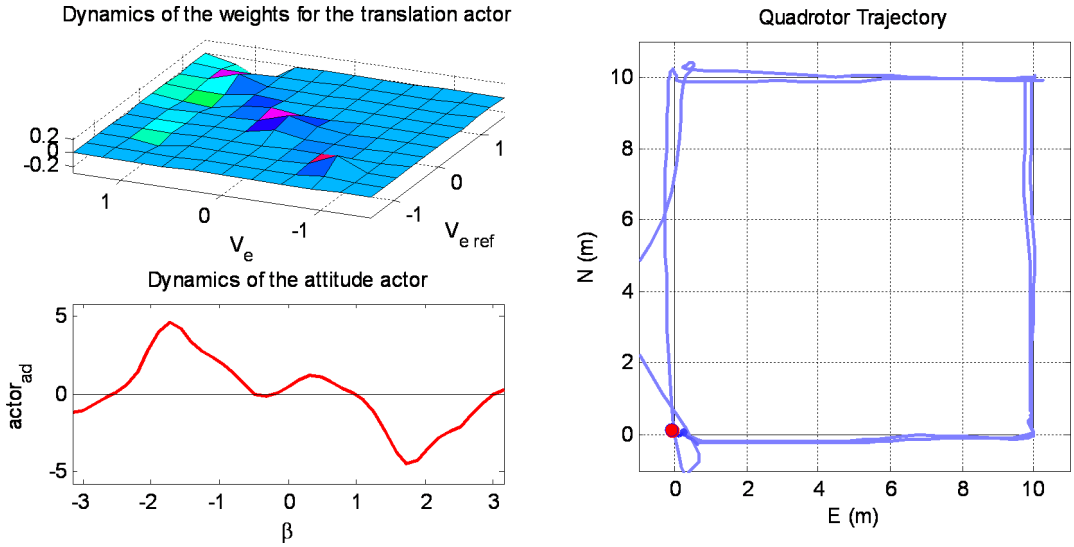


Figure 45. Adaptation to both wind and weight imbalance

5.10. Flight Test

A flight test was done in windy weather. The average wind speed was 12 km/h with wind gusts of 14 km/h. The Simulink model for the controller was the same as the one used for the simulations, but the main quadrotor model block was replaced by a block that communicates directly to the hardware in real-time. The neural network weights for the actors were initialized to known stable PD controllers used in the flight test from Chapter 4. The plots in the next figures show what has been learned in addition to the initial PD configuration.

The reference flight trajectory was the same square with 10 meter sides. As seen in Figure 46, the pitch angle has strong oscillations immediately after take-off. The controller is able to adapt to the wind conditions in 40 seconds and the oscillations are reduced significantly.

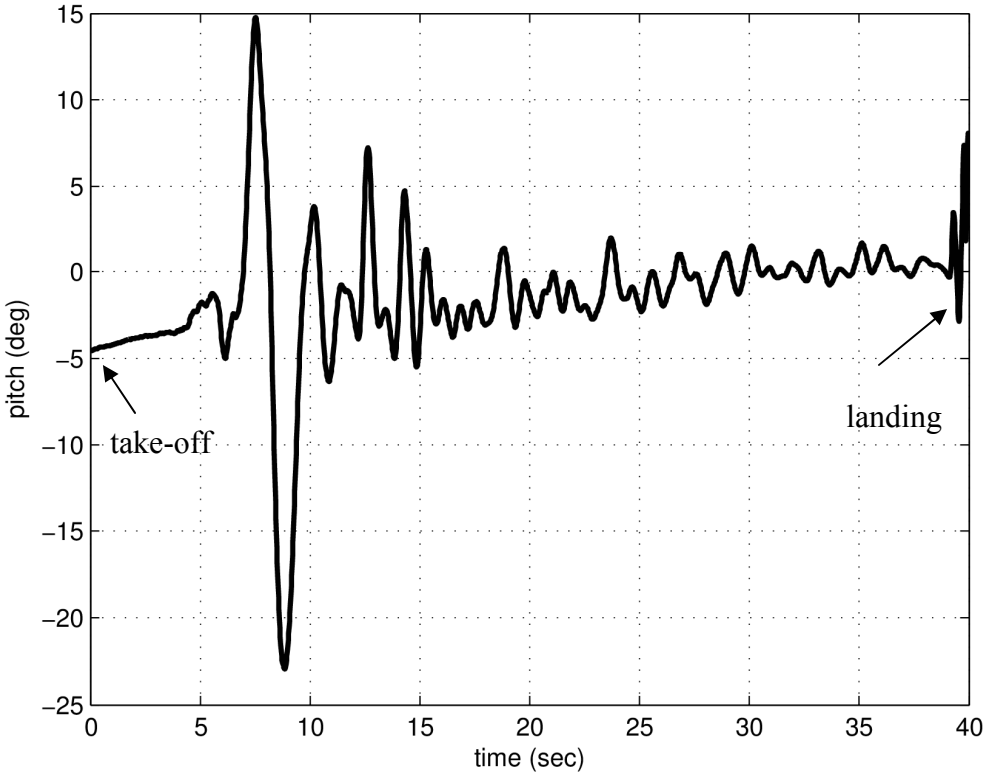


Figure 46. Pitch angle during flight

Although the attitude subsystem has adapted properly (Figure 47), the translation subsystem failed completely. The trajectory of the quadrotor (Figure 48) did not follow the square-shaped reference but instead has followed the wind direction (blowing from N-NW). An explanation can be obtained by looking at the weights of the translation actors with projections for the North and the East directions (Figures 49–50). It can be seen that adaptation only took place at the edge of the intervals allowed for the translation velocities. The wind speed was much higher than the 1 m/s limits allowed for the quadrotor velocities. Once the quadrotor went out of those limits by being carried by the wind, the adaptation for the attitude actor became impossible. This shows

the importance of designing the neural networks for the actors with an extended input range for robustness.

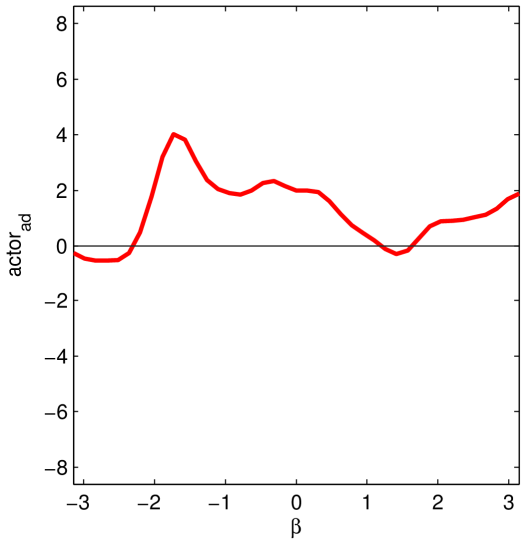


Figure 47. Attitude actor weights

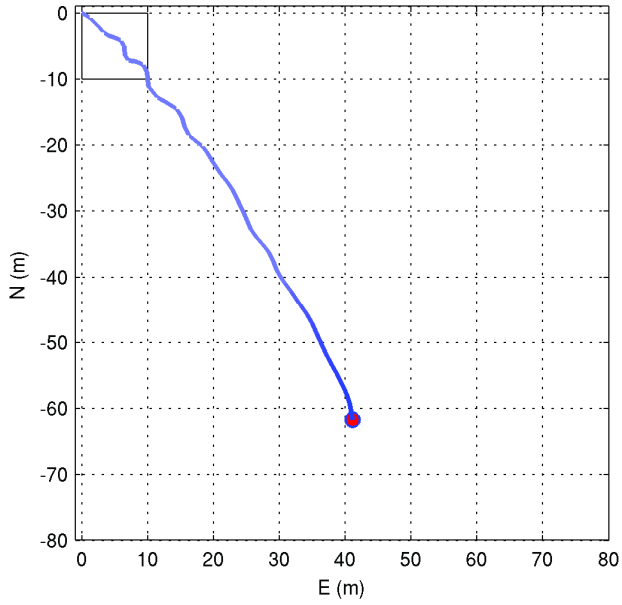


Figure 48. Trajectory of the quadrotor

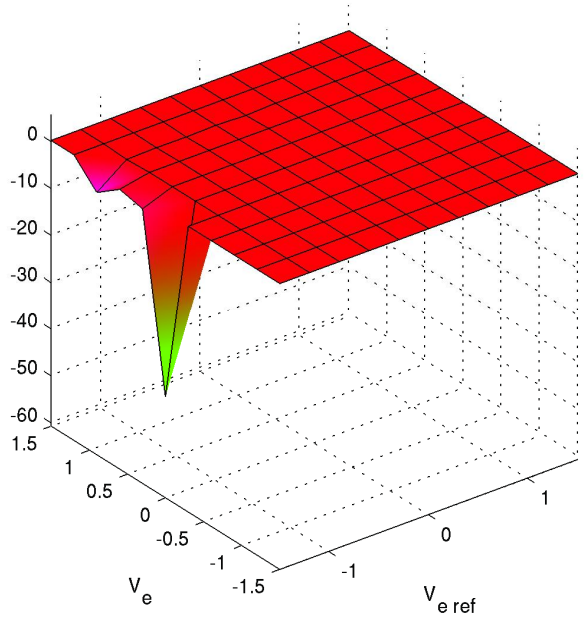


Figure 49. Translation actor weights for the North direction

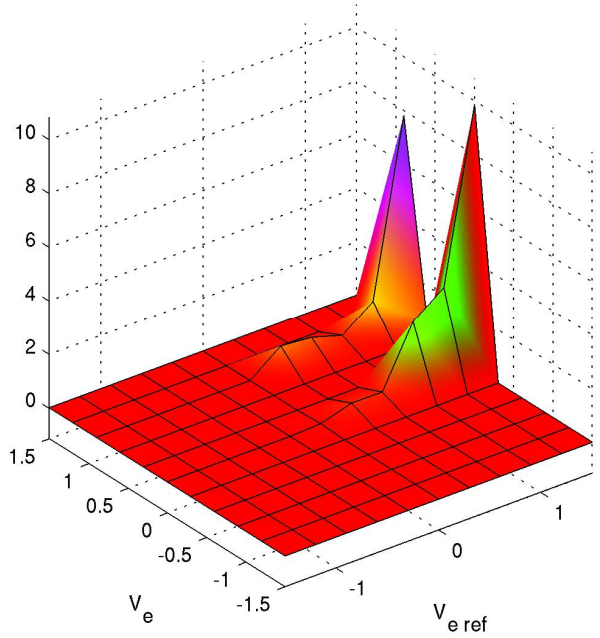


Figure 50. Translation actor weights for the East direction

5.11. Conclusion

Model-based ADP algorithms do not converge to the optimal solution when the model is not fully known or when its parameters deviate from the nominal values. In this case model-free algorithms are needed. Unfortunately, model-free ADP algorithms can not be used on complex systems directly. They can be considered only after enough information about the system and the environment has been acquired and stored in the neural networks using other methods. They can further refine the learning, but are not suitable for coarse learning from the beginning. A future implementation of a control algorithm for a quadrotor that allows the model to change and still converge to the optimum control policy would have to follow these steps:

- Model the system and the environment with what precision is possible.
- Use on-line or off-line model-based ADP to train the Actor and Critic NN. The NN should be able to store information for any operating point, not only for the nominal operating point.
- Use the on-line model-free ADP algorithm with the pre-trained NN. Only local exploration would be sufficient for reaching optimality even if the model starts departing from nominal values.

This chapter presented the development of on-line ADP reinforcement learning algorithms in suitable format for implementation on a quadrotor platform. A formulation for reinforcement learning was developed that consists of focused learning on subsystems of the quadrotor, with each subsystem having its own control actor yet all subsystems having a globally defined utility function. An objective was to formulate

a realistic structure for ADP for practical implementation that also relates to standard control system structures such as PID and nonlinear algorithms.

CHAPTER 6

DISTRIBUTED CONTROL OF MOBILE ROBOTS

6.1. Nomenclature

Ω_L, Ω_R	angular velocities of the wheels (rad/s)
d	wheels diameter (m)
l	distance between wheels (m)
x, y	position coordinates in the inertial frame (m)
θ	heading angle (rad)
V	velocity in the body frame (m/s)
ω	angular velocity (rad/s)
$G(V, E)$	communication graph
d_i	in-degree
d_i^o	out-degree
t_{ij}	trust of node i about node j
p_{ij}	predictability by node i of the node j behavior
$\mathbf{x}_{ij}$	state info about node j held by node i

h_i number of hops from node i to the leader

R_i set of nodes that request connectivity to node i

6.2. Introduction

Motion path planning is the process of finding a continuous path from an initial position to a prescribed final position (goal) without collision. Artificial potential field methods for obstacle avoidance have gained increased popularity among researchers in the field of mobile robots. The idea of imaginary forces acting on a robot has been suggested by Andrews and Hogan [11] and Khatib [12]. It is generally easy to develop centralized algorithms that apply potential field methods for all robots based on complete information about their state variables. This requirement is not usually achieved in practice. For many applications, distributed control algorithms are required to operate on each mobile robot under conditions of restricted communication channels between the robots. This chapter explains the basic idea behind the potential field path planning for a formation of mobile robots that communicate over a graph structure. Due to the dynamic environment and the limited communication resources, the robots play a network formation game [43] in order to be able to collect information from the most trusted nodes and to avoid un-trusted agents.

The communication graph structure switches at every sample step in order to maximize a utility function for each node. This switching behavior guarantees that information is obtained from most neighbors at successive moments in time because the resources don't allow for it to be obtained all at once. Estimation of the behavior of the

neighbors done by each node compensates somehow for the lack of very recent information.

Although the algorithm is developed for ground robots, the same ideas can be used to control a formation of quadrotors in two or in three dimensions.

6.3. Graph Building Algorithm

The structure of the communication links between the robots is described by using a directed graph (Figure 51). A graph is a pair $G = (V, E)$ with V a nonempty set of M nodes and a set of edges $E \subseteq V \times V$. The edges are represented by an adjacency matrix $A = [a_{ij}]$ with $a_{ij} = 1$ if the node j is outgoing and the node i is accepting data from node j, and $a = 0$ otherwise. The in-degree d_i of a node i is the number of edges that have node i as a destination. The out-degree d_i^o is the number of edges that have node i as a source. The in and out-degree are restricted to have the maximum values $d_{\max}$ and respectively $d_{\max}^o$. In addition, for each node i a set of visible nodes is defined:

$$V_i^V = \left\{ j \left| \sqrt{\left(x_i - x_j\right)^2 + \left(y_i - y_j\right)^2} < r_{vis}, j \neq i \right. \right\} \tag{177}$$

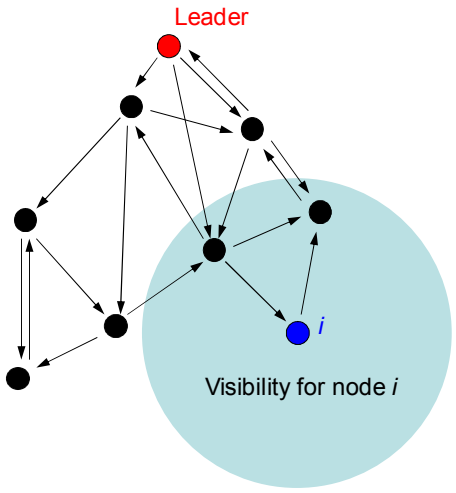


Figure 51. Graph structure for the robot formation

The graph building algorithm and the control algorithm work in discrete time. The node dynamics in discrete time is

$$\mathbf{x}_i(k+1) = \mathbf{x}_i(k) + Tf_i\left(\mathbf{x}_i(k), \mathbf{u}_i(k)\right) \tag{178}$$

In addition, each node maintains a list of extra states relative to the other nodes: trust, predictability, and number of hops to the formation leader. Their dynamics is given below:

For every $p \in (1,...,M)$ we have

$$t_{ip}\left(k+1\right) = sat\left[\begin{array}{l} \dfrac{1}{1+\sum_{j\in N_i}\delta\left(t_{ij}(k)\right)t_{ij}(k)} \cdot \\ \cdot\left(t_{ip}(k)+\sum_{j\in N_i}\delta\left(t_{ij}(k)\right)t_{ij}(k)t_{jp}(k)\right)+ \\ +\lambda_1-\lambda_2\cdot e^{-\frac{r_{ip}^2}{2r_{\min}^2}} \end{array}\right] \tag{179}$$

where $-1 < t_{ij} < 1$, $\delta()$ is the Heaviside step function and $sat(x)$ clips x between -1 and 1 which means complete distrust, and complete trust respectively. The trust propagation algorithm uses a voting mechanism. The votes for node p coming from the connected neighbors are weighted through the trust value that node i has for those neighbors. Trust from other nodes j is considered only if node i has a positive trust about them ($t_{ij} > 0$). The λ_1 constant causes the trust to slowly increase over time if no incidents take place regarding the minimum distance between nodes.

The predictability is also limited to the $[-1,1]$ interval.

$$p_{ip}(k+1) = sat\left[p_{ip}(k) - \lambda_3 \right] \text{ if } p \notin N_i \tag{180}$$

and

$$p_{ip}(k+1) = sat\left[p_{ip}(k) + \lambda_4 - \lambda_5 \left\| \hat{\mathbf{x}}_{ip}(k) - \mathbf{x}_{ip}(k) \right\| \right] \tag{181}$$

if the data from node p is available at the current time step.

If no data is available from node p, the value associated to its predictability decreases linearly until it hits -1 or new data becomes available. The λ_4 constant causes the predictability to increase if there are only small differences between the behavior of node p and its estimate made by node i. A negative value for the predictability decreases the trust in that node until trust becomes strongly negative. This causes the utility function to ask for data from unpredictable nodes periodically.

The information about the number of hops from the current node i to the formation leader helps in building networks with a better connectivity and shorter links:

$$h_i = \min_{j \in N_i} \left\{ h_j \right\} + 1. \tag{182}$$

Node i keeps an estimate for the states of all the other nodes. The control law is generated using potential fields:

$$\hat{\mathbf{x}}_{ip}(k+1) = \hat{\mathbf{x}}_{ip}(k) + Tf_p\left(\hat{\mathbf{x}}_{ip}(k), \mathbf{u}_{ip}(k)\right) \text{ if } p \notin N_i \tag{183}$$

and

$$\hat{\mathbf{x}}_{ip}(k+1) = \mathbf{x}_{ip}(k) + Tf_p\left(\mathbf{x}_{ip}(k), \mathbf{u}_{ip}(k)\right) \text{ if } p \in N_i \tag{184}$$

where

$$F_{x,ip} = \sum_{\substack{j=1 \\ j \neq p}}^{M} \frac{1}{2}\left(t_{ij} + 1\right) K_2\left(\hat{x}_{ip} - \hat{x}_{ij}\right) - K_3 \frac{\hat{x}_{ip} - \hat{x}_{ij}}{r_{pj}} e^{-\frac{r_{pj}^2}{2\left[r_{robot} + K_4\left(1 - t_{pj}\right)\right]^2}} \tag{185}$$

$$F_{y,ip} = \sum_{\substack{j=1 \\ j \neq p}}^{M} \frac{1}{2}\left(t_{ij} + 1\right) K_2\left(\hat{y}_{ip} - \hat{y}_{ij}\right) - K_3 \frac{\hat{y}_{ip} - \hat{y}_{ij}}{r_{pj}} e^{-\frac{r_{pj}^2}{2\left[r_{robot} + K_4\left(1 - t_{pj}\right)\right]^2}} \tag{186}$$

$$r_{pj} = \sqrt{\left(\hat{x}_{ip} - \hat{x}_{ij}\right)^2 + \left(\hat{y}_{ip} - \hat{y}_{ij}\right)^2} \tag{187}$$

$$\alpha_{ip} = atan2\left(F_{y,ip}, F_{x,ip}\right) \tag{188}$$

$$u_{ip}(k) = \begin{bmatrix} V_{ip} \\ \omega_{ip} \end{bmatrix} = \begin{bmatrix} K_V \sqrt{F_x^2 + F_y^2} \\ K_\omega\left(\alpha_{ip} - \hat{\theta}_{ip}\right) \end{bmatrix} \tag{189}$$

The attraction term from the potential field forces depends on the trust values. If there is complete distrust, the attractive potential disappears and the repulsive potential is the strongest. That way, nodes stay away from un-trusted elements and can get closer to the ones that are trusted.

The core of the distributed graph construction consists of the definition of two utility functions that capture the benefits and costs for a given strategy s_i. One utility function corresponds to the incoming connection and the other corresponds to the outgoing connections. The set of possible strategies is

$$S_i = \left\{ ji \middle| j \in R_i \right\} \times \left\{ ij \middle| j \in V_i^V - \{i\} \right\} \tag{190}$$

The algorithm searches for the strategy

$$s_i^* = \left(a_i^*, b_i^* \right) \in S_i \tag{191}$$

that generates the highest value for the utility functions. The set a_i^* represents the best choice of nodes to whom to send data and b_i^* represents the best choice of nodes to be asked to provide data.

The utility functions for node i are

$$U_i^a(G) = -C\left(d_i^o\right) + \beta \sum_{j \in N_i^0} E\left(d_j\right) \tag{192}$$

$$U_i^b(G) = -C\left(d_i\right) + \alpha \sum_{j \in N_i} \left| t_{ij} \right| - \gamma \sum_{j \in N_i} \delta\left(t_{ij}\right) t_{ij} p_{ij} - \rho \sum_{j \in N_i} h_j \tag{193}$$

where

$$C\left(d_i\right) = -c \cdot \log\left[1 - \left(\frac{d_i}{d_{\max}} \right)^2 \right] \tag{194}$$

$$C\left(d_i^o\right) = -c^o \cdot \log\left[1 - \left(\frac{d_i^o}{d_{\max}^o} \right)^2 \right] \tag{195}$$

and

$$E\left(d_j\right) = -c^e \cdot \log\left[1 - \left(\frac{d_{\max} - d_j}{d_{\max} + 1}\right)^2\right] \tag{196}$$

are log barrier functions.

The first term in the utility functions penalizes a large number of incoming or outgoing connections. It also guarantees that there are no connections beyond the allowed maximum number $d_{\max}$ and $d^o_{\max}$. The second term in $U_i^a\left(G\right)$ gives priority to the acceptance of outgoing connections to nodes that have a small number of inputs. This behavior is included to prevent letting nodes without any input at all. The β_i constant can be thought of as a value describing the altruism of node i. The second term in $U_i^b\left(G\right)$ adds a preference for receiving connections from nodes that are either trusted more or distrusted more. The latter case is needed because nodes that are distrusted have to be closely monitored to prevent collisions. The third term discourages the creation of links to nodes that are both trusted and predictable. The states of these nodes should already be well estimated locally and there is no need to ask for information from them too often. The last term penalizes long graph structures that can have nodes far away from the leader. Such structures would introduce long propagation delays and would not provide an adequate behavior for the formation.

The c, c^o, c^e, α, β, γ and ρ constants are carefully chosen in order to make the utility function feasible.

The graph formation and the control algorithm is the following (Figure 52):

Initial State

All the nodes start connected in a tree structure with the formation leader at the root. All the nodes are initialized with trust values $t_{ij} = 1$ for all the other agents. The number of hops to the leader is calculated for each node. The leader receives a value of 0. The predictability values are set to zero.

Discrete-time loop. At time k, each node:

Receives data on the incoming links and sends data on the outgoing links. The variables that are transferred by node i are the following: $\mathbf{x}_i(k)$, $t_{ij}(k)$, $h_i(k)$ with $j \in (1,...,M)$.

Calculates $\hat{\mathbf{x}}_{ij}(k+1)$, $t_{ij}(k+1)$ and $h_i(k+1)$ based on the data it has received at time k.

Calculates its node visibility vector based on $\hat{\mathbf{x}}_{ij}(k+1)$.

Finds the best nodes a_i^* to receive data from. Makes a request to the nodes in b_i^* to provide data to it at the next time step $k+1$ and provides them with d_i. The data requests that are received are stored in R_i. Decides to which nodes a_i^* to send data to at time step $k+1$.

The loop repeats at every sample time.

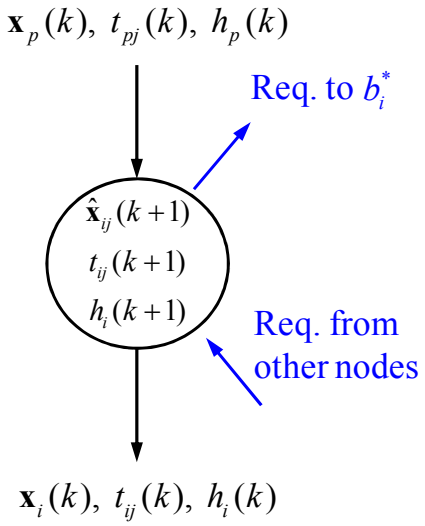


Figure 52. Data flow within the graph building algorithm

6.4. Implementation Results

The potential field used to generate commands for the robots is designed using the K_2, K_3, K_4 and r_{robot} parameters in equations 185–186. K_2 controls the strength of the attractive field. K_3 determines the height of the repulsive field. K_4 controls how wide the repulsive field becomes when there is no trust. The next figures show the configuration of the potential field for three levels of trust in a robot.

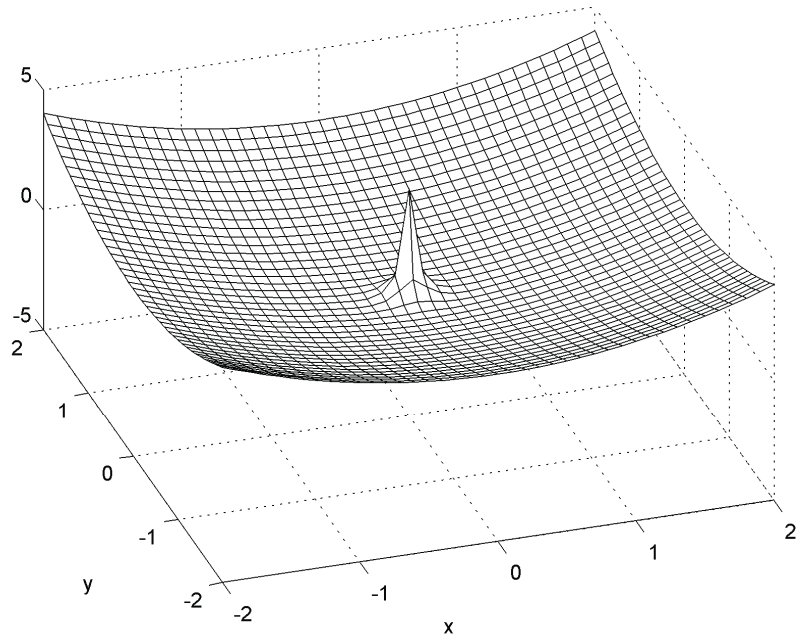


Figure 53. Potential field for a node that is completely trusted

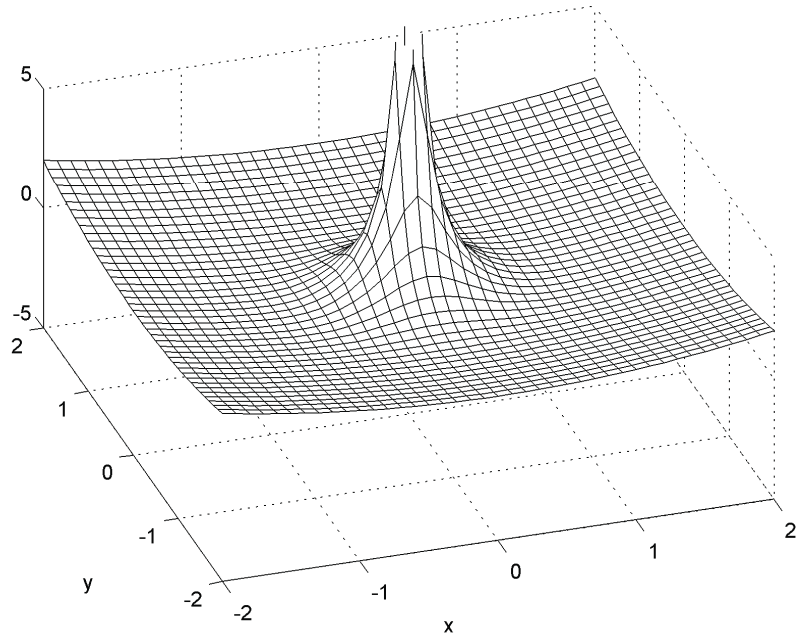


Figure 54. Potential field for a node that has zero trust

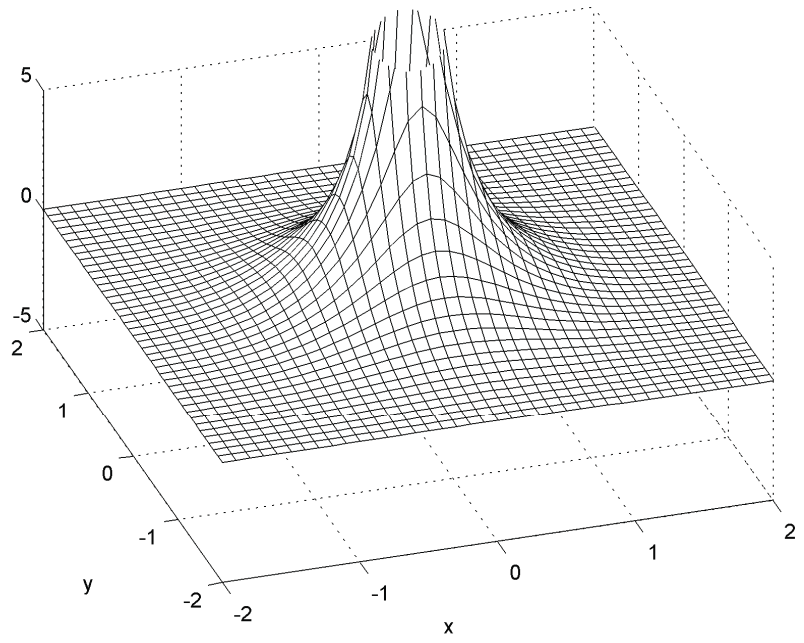


Figure 55. Potential field for a node that is completely distrusted

Robot 1 is forced to travel diagonally from the lower-left to the upper-right corner. The other robots are placed in its path. The initial configuration is shown in Figure 56. When robot 1 gets too close to robot 2, the trust that robot 2 has about robot 1 decreases (Figure 59). The collision is avoided and robot 1 continues its path. Since there is no danger of a collision to robot 2, the trust that robot 2 has about robot 1 starts to increase again. During all this time, robot 2 communicates with various neighbors and with robot 1. Robot 2 asks for messages from robot 1 more often when the trust is smaller.

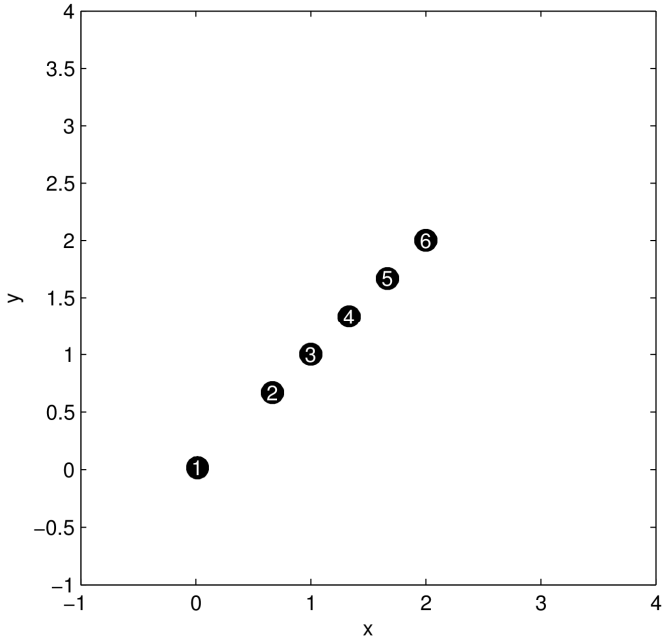


Figure 56. Initial configuration of the robots

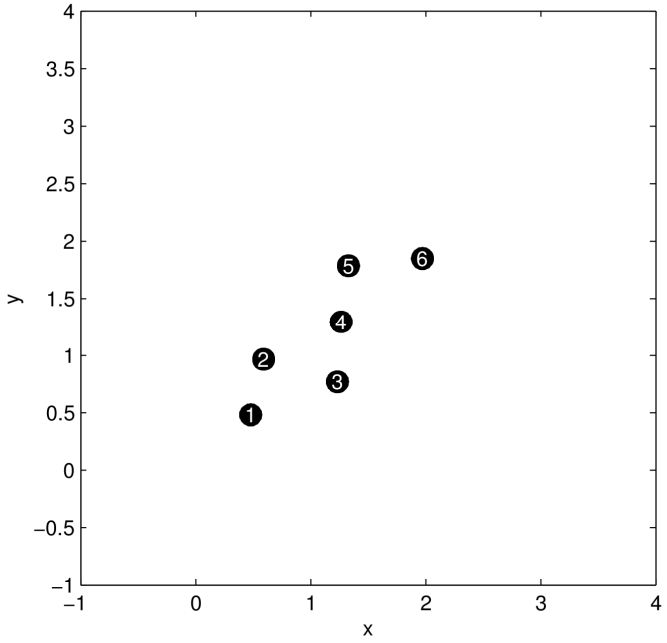


Figure 57. Node 1 gets too close to node 2

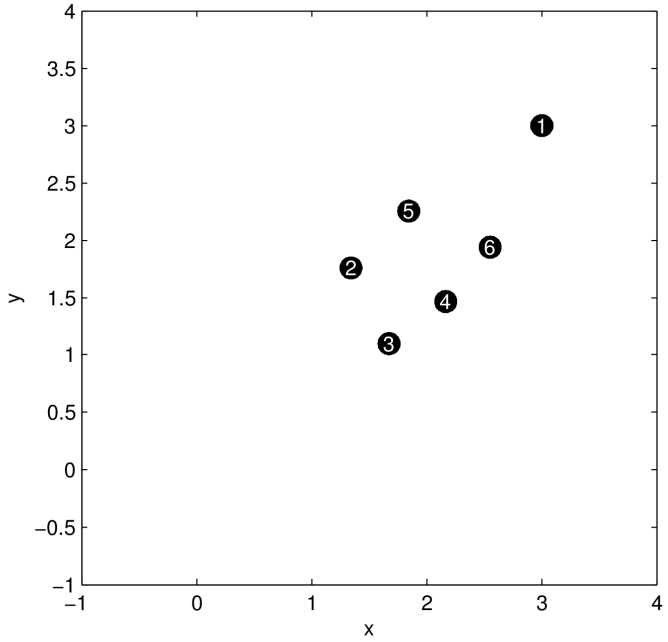


Figure 58. Final configuration of the robots

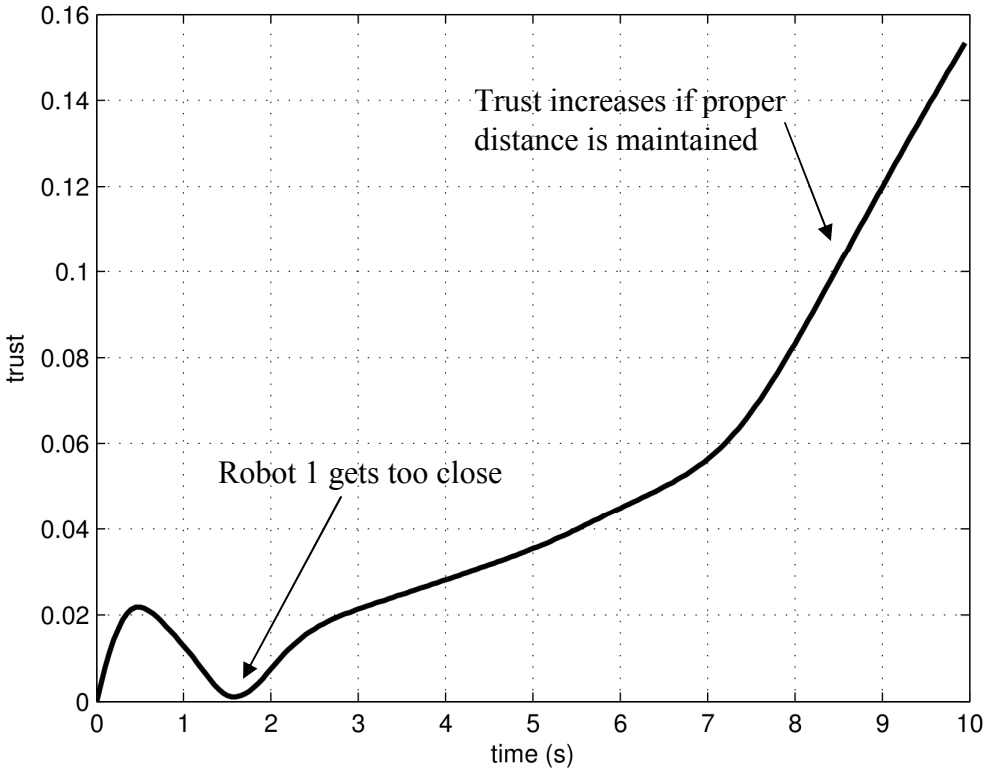


Figure 59. The trust that robot 2 has in robot 1

6.5. Conclusion

This chapter presented an online, real-time, decentralized algorithm for graph building for a formation of mobile robots with limited communication resources. The algorithm uses additional state variables such as trust and predictability stored by every node in order to compensate for the lack of communication links between all the robots. The communication graph structure switches at every sample step in order to maximize a utility function for each node. This switching behavior guarantees that information is obtained from most neighbors at successive moments in time because the resources don't allow for it to be obtained all at once. Estimation of the behavior of the neighbors done by each node compensates for the lack of very recent information. The algorithm is very well suited to real-world applications in hostile environments where the amount of information that can be exchanged between nodes has to be severely restricted.

Future research is needed to find ways to automatically optimize the tuning parameters in the utility function for different formation configurations and different communication parameters.

CHAPTER 7

CONCLUSION

7.1. Conclusion

There is currently a dichotomy between optimal control and adaptive control. Adaptive control algorithms learn online and give controllers with guaranteed performance for unknown systems. On the other hand, optimal control design is performed off line and requires full knowledge of the system dynamics. The current work develops an optimal adaptive controller that solves approximately the optimal control problem for nonlinear systems online in real-time. This solution requires some knowledge of the plant structure and dynamics.

Research topics in UAV control become more and more complex. The industry and the Armed Forces are reluctant to accept complicated theoretical solutions without a practical implementation to prove their applicability. The previous chapters have presented the control structures, the models and some algorithms that can be used to make the robots autonomous, to allow the human pilots to provide only low-bandwidth, high-level commands without worrying about the internal dynamics of the vehicles they control, and to allow the robots to adapt to and to gather knowledge about the environment.

The major contribution of this work is the application of ADP algorithms to complex robotic platforms that have to accomplish their missions in unstructured environments where their success depends on their ability to learn and to adapt to unforeseen conditions.

The scale of the problem seems too big for standard ADP algorithms that are usually applied to systems with two or three states. Even more, we try to solve a trajectory tracking problem, not the regulation problem. A compromise had to be made. Instead of having overall optimality, the control algorithm is split around three smaller loops with similar behavior: translation, attitude and motor/propeller control. A global critic is maintained, but the overall behavior only converges to some local optimum.

Learning is focused by pre-processing the inputs to the neural networks in order to reduce the number of input dimensions and to provide significant and meaningful data that is stored in the same place for equivalent contexts. The value function and the policy that correspond to the infinite number of combinations of state variable values and possible commands have to be stored using a finite number of parameters. The coding of these two functions is made using function approximation with a modified version of RBF neurons. Due to their local effect on the approximation, the RBF neurons are best suited to hold information that corresponds to training data generated only around the current operating point, which is what one can obtain by following a normal trajectory without extensive exploration.

Autonomy of a single robotic platform is usually not enough for the complex missions that involve a large number of robots. Intelligent, distributed formation control

over graph structures when limited communication resources are available is a requirement in such cases. Inspired by nature, a trust variable is introduced. It stores information about the other nodes that can be used in place of costly negotiations between them. The control algorithm uses trust to generate the commands in order to avoid misbehaving nodes and to maintain a formation. At the same time the trust dynamics is affected by the behavior of the neighboring nodes and by their own estimation of trust through a local voting protocol.

7.2. Future Research Directions

During the 20th century, control theory has played a major role in Engineering and Technology. The 21st century will certainly see major advancements in this field. Classical control will be replaced with theory that can be applied not only to relatively simple, conventional installations, but also to more general systems from Biology, Sociology, Economy, Internet and many others. A list of nature-inspired concepts from the present work can be easily enumerated: neural networks, fuzzy logic, the subsumption architecture, potential fields, reinforcement learning, focalization and localization of learning, restricted communication, trust. The next major development in control theory will be based on the way the information is processed to produce knowledge. ADP is a first step in this direction. Data mining algorithms can make it more efficient and more robust to the various sources of noise found in the real-world environment.

REFERENCES

[1] L.A. Zadeh, Outline of a new approach to the analysis of complex systems and decision processes. IEEE Transactions on Systems, Man and Cybernetics, Vol. SMC-3, pp. 28-44, 1973

[2] L.X. Wang, Fuzzy systems are universal approximators. Proceedings of the 1st IEEE Conference on Fuzzy Systems, San Diego, CA, pp. 1163-1170, 1992

[3] J. Park, I.W. Sandberg, Universal approximation using radial-basis-function networks. Neural Computation, Vol. 3, pp. 246-257, 1991

[4] P.J. Werbos, Backpropagation: basics and new developments. The handbook of brain theory and neural networks, MIT Press, Cambridge, MA, USA, pp. 134-139, 1998

[5] R.E. Burke, Spinal Reflexes. Encyclopedia of Life Sciences, John Wiley & Sons, 2001

[6] A. Saffiotti, The Uses of Fuzzy Logic in Autonomous Robotics: a catalogue raisonne. In Soft Computing, I(4), Springer-Verlag, pp 180-197, 1997

[7] J.P. Donoghue, J.N. Sanes, Motor System Organization. Encyclopedia of Life Sciences, John Wiley & Sons, 2001

[8] R.A. Brooks, A Robust Layered Control System For A Mobile Robot. IEEE Journal of Robotics and Automation, RA-2, pp. 14-23, 1986

[9] G.N. Saridis, H.E. Stephanou, A hierarchical approach to the control of a prosthetic arm. IEEE Transactions on Systems, Man and Cybernetics, Vol. SMC-7, no. 6, pp. 407-420, 1977

[10] D.R. Lefebvre, G.N. Saridis, A Computer Architecture for Intelligent Machines. Intelligent Robotic Systems for Space Exploration, pp. 31-43, 1991

[11] J.R. Andrews, N. Hogan, Impedance Control as a Framework for Implementing Obstacle Avoidance in a Manipulator. Control of Manufacturing Processes and Robotic Systems (D. E. Hardt and W. Book, eds.), ASME, Boston, pp. 243-251, 1983

[12] O. Khatib, Real-time obstacle avoidance for manipulators and mobile robots. IEEE Int. Conf. Robotics and Automation, St. Louis, MO, pp. 500-505, 1985

[13] E.H. Ruspini, Truth as utility: a conceptual synthesis. Procs. of the 7th Conf. on Uncertainty in Artificial Intelligence, Los Angeles, CA, 1991

[14] J. Yen, N. Pfluger, A Fuzzy Logic Based Extension to Payton and Rosenblatt's Command Fusion Method for Mobile Robot Navigation. IEEE Transactions on Systems, Man and Cybernetics, Vol. 25(6), pp. 971-978, 1995

[15] E.H. Ruspini, Fuzzy logic in the Flakey robot. Procs. of the Int. Conf. on Fuzzy Logic and Neural Networks (IIZUKA), pp 767-770, Iizuka, Japan, 1990

[16] A. Saffiotti, K. Konolige, E.H. Ruspini, A multivalued-logic approach to integrating planning and control. Artificial Intelligence, Vol. 76, Issue 1-2, pp. 481-526, 1995

[17] A. Saffiotti, E.H. Ruspini, K. Konolige, Blending reactivity and goal-directedness in a fuzzy controller. Proceedings of the IEEE Int. Conf. on Fuzzy Systems, San Francisco, California, pp. 134-139, 1993

[18] L. Ojeda, J. Borenstein, FLEXnav: Fuzzy Logic Expert Rule-based Position Estimation for Mobile Robots on Rugged Terrain. Proceedings of the 2002 IEEE International Conference on Robotics and Automation, Washington DC, USA, pp. 317-322, 2002

[19] F.L. Lewis, L. Xie, and D. Popa, Optimal & Robust Estimation: With an Introduction to Stochastic Control Theory, CRC Press, Boca Raton, 2007. Second Edition.

[20] M.E. Dreier, Introduction to Helicopter and Tiltrotor Flight Simulation. AIAA Education Series, AIAA, 2007

[21] J.C.K. Chou, Quaternion kinematic and dynamic equations. IEEE Transactions on Robotics and Automation, 8(1), 53–64, 1992.

[22] S. Park, D.H. Won, M.S. Kang, T.J. Kim, H.G. Lee, S.J. Kwon, RIC (robust internal-loop compensator) based flight control of a quad-rotor type UAV. IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 3542–3547, Aug. 2005.

[23] P. Castillo, A. Dzul, R. Lozano, Real-time stabilization and tracking of a four-rotor mini rotorcraft. IEEE Transactions on Control Systems Technology, vol. 12, no. 4, pp. 510–516, July 2004.

[24] M. Valenti, B. Bethke, G. Fiore, J.P. How and E. Feron, Indoor Multi-Vehicle Flight Testbed for Fault Detection, Isolation, and Recovery. AIAA Guidance, Navigation and Control Conference, Aug. 2006.

[25] N. Guenard, T. Hamel, V. Moreau, Dynamic modeling and intuitive control strategy for an X4-flyer. International Conference on Control and Automation, vol. 1, pp. 141–146, June 2005.

[26] S. Bouabdallah and R. Siegwart, Full control of a quadrotor. IEEE/RSJ International Conference on Intelligent Robots and Systems, pp.153–158, Oct. 2007.

[27] G.M. Hoffmann, H. Huang, S.L. Wasl, and C.J. Tomlin, Quadrotor helicopter flight dynamics and control: Theory and experiment. AIAA Guidance, Navigation, and Control Conference, 2007.

[28] M.D. Shuster, S.D. Oh, Three-Axis Attitude Determination from Vector Observations. Journal of Guidance and Control, vol. 4, no. 1, pp. 70–77, January–February 1981.

[29] S. Gupta, Linear quaternion equations with application to spacecraft attitude propagation. IEEE Aerospace Conference, vol. 1, pp. 69–76, March 1998.

[30] C. Nicol, C.J.B. Macnab, and A. Ramirez-Serrano, Robust Neural Network Control of a quadrotor Helicopter. IEEE Canadian Conference on Electrical and Computer Engineering, pp. 1233–1238, 2008.

[31] T. Dierks and S. Jagannathan, Output Feedback Control of a Quadrotor UAV Using Neural Networks. IEEE Transactions on Neural Networks, Vol. 21, No. 1, pp. 50–66, 2010.

[32] R.J. Leake and R.W. Liu, Construction of suboptimal control sequences. J. SIAM Contr., Vol 5, No. 1, pp. 54–63, 1967.

[33] R. Beard, G. Saridis, and J. Wen, Approximate solutions to the time-invariant Hamilton-Jacobi-Bellman equation. Automatica, Vol. 33, No. 12, pp. 2159–2177, 1997.

[34] F.L. Lewis and V. Syrmos, Optimal Control. 2nd ed. New York: Wiley, 1995.

[35] S. Marsili-Libelli, Optimal design of PID regulators. International Journal of Control, 33(4), pp. 601–616, 1981.

[36] Y.M. Park, M.S. Choi, and K.Y. Lee, An optimal tracking neuro-controller for nonlinear dynamic systems. IEEE Transactions on Neural Networks: Vol 7, Issue 5, pp. 1099–1110, 1996.

[37] H. Zhang, Q. Wei, and Y. Luo, A Novel Infinite-Time Optimal Tracking Control Scheme for a Class of Discrete-Time Nonlinear Systems via the Greedy HDP Iteration Algorithm. IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics, Vol 38, Issue 4, pp. 937–942, 2008.

[38] R.E. Bellman, Dynamic Programming. Princeton, NJ: Princeton Univ. Press, 1957.

[39] R.S. Sutton and A.G. Barto, Reinforcement Learning: An Introduction. Adaptive Computation and Machine Learning, MIT Press, 1998.

[40] C. Watkins, Learning from delayed rewards. Ph.D. thesis, Cambridge University, Cambridge, England, 1989.

[41] C. Watkins and P. Dayan, Q-learning. Machine Learning, Vol 8, pp. 279–292, 1992.

[42] Y. Li, N. Sundararajan, and P. Saratchandran, Analysis of minimal radial basis function network algorithm for real-time identification of nonlinear dynamic systems. IEE Proceedings–Control Theory and Applications, Vol 147, Issue 4, pp. 476–484, 2000.

[43] W. Saad, Han Zhu, M. Debbah, A. Hjorungnes, T. Basar, Coalitional game theory for communication networks. Signal Processing Magazine, IEEE , Vol. 26, No.5, pp. 77-97, 2009.

[44] J.S. Baras and T. Jiang, Cooperation, Trust and Games in Wireless Networks. In Advances in Control, Communication Networks, and Transportation Systems: In Honor of Pravin Varaiya, E.H. Abed (Ed.), Systems and Control: Foundations and Applications Series, Birkhauser, Boston, 2005.

[45] D.C. Arney and E. Peterson, Cooperation in Social Networks: Communication, Trust, and Selflessness. Proceedings of the 2008 Army Science Conference, 2008.

BIOGRAPHICAL INFORMATION

Emanuel Stingu received his Bachelor of Engineering degree in Automatic Control from the Faculty of Automatic Control and Computer Engineering, Technical University of Iasi, Romania, in 2006. In the same year he started his Masters in Electrical Engineering at the University of Texas at Arlington under the supervision of Dr. Frank Lewis. He received his Master of Science degree in 2008 and continued with the Ph.D. program. He has been working as a Graduate Research Assistant at the Automation & Robotics Research Institute.

He showed interest in electronics and computer programming ever since he was in the 5th grade in elementary school, winning top places in Electronics and Physics competitions at the national level in Romania. His experience allows him to easily combine the control theory with the requirements of practical systems. His interests are mostly in nonlinear control, including modeling, system identification and robust/adaptive control, and multi-agent systems.