

# An Embedded Controller for Quad-Rotor Flying Robots Running Distributed Algorithms

by

Brian John Julian

B.S., Cornell University (2005)

Submitted to the Department of Electrical Engineering and Computer Science

in partial fulfillment of the requirements for the degree of

Master of Science

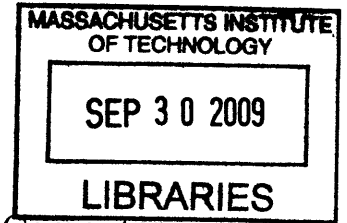
at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2009

© Brian John Julian, MMIX. All rights reserved.

The author hereby grants to MIT and MIT Lincoln Laboratory permission to reproduce and distribute publicly paper and electronic copies of this thesis document in whole or in part.



**ARCHIVES**

Author .....  
Department of Electrical Engineering and Computer Science  
August 20, 2009

Certified by .....  
Daniela L. Rus  
Associate Professor of Electrical Engineering and Computer Science  
Thesis Supervisor

Accepted by .....  
Terry P. Orlando  
Chairman, Department Committee on Graduate Students

This work is sponsored by the Department of the Air Force under Air Force contract number FA8721-05-C-0002. Opinions, interpretations, conclusions, and recommendations are those of the author and are not necessarily endorsed by the United States Government.



# **An Embedded Controller for Quad-Rotor Flying Robots Running Distributed Algorithms**

by

Brian John Julian

Submitted to the Department of Electrical Engineering and Computer Science  
on August 20, 2009, in partial fulfillment of the  
requirements for the degree of  
Master of Science

## **Abstract**

Multiple collaborating quad-rotor flying robots are useful in a broad range of applications, from surveillance with onboard cameras to reconfiguration of wireless networks. For these applications, it is often advantageous to have the robot team be a distributed system. In this thesis, an embedded controller capable of running distributed algorithms is presented for the quad-rotor flying robot.

The robot platform is first characterized to help guide the design of the embedded control module. These modules are fabricated and tested on the quad-rotor flying robots in both indoor and outdoor environments. To propagate state estimates throughout the robot team, a location-based multi-hop algorithm is proposed. Network limitations, such as sub-optimal bandwidth and finite communication range, are implemented in hardware-in-the-loop simulations to determine system performance. A novel coverage algorithm for multiple hovering robots with downward facing cameras is then demonstrated on the embedded controller. The results from numerous indoor and outdoor experiments are discussed.

Thesis Supervisor: Daniela L. Rus

Title: Associate Professor of Electrical Engineering and Computer Science





## Acknowledgments

This research would not have been possible without the support of those around me. I would like to thank my advisor Daniela Rus, whose steadfast confidence in my work kept me excited and motivated through the toughest challenges. I am indebted to all the members of the Distributed Robotics Laboratory for their willingness to endure late nights even if not for their own projects. In particular, Mac Schwager and Michael Angermann have my highest respect for their guidance and good humor. Carrick Detweiler and Iuliu Vasilescu also deserve special recognition for the many valuable discussions and debugging sessions.

I would also like to thank MIT Lincoln Laboratory for graciously funding my graduate education. Special thanks goes to:

- My group leaders Craig Perini and Steve Forman, for providing critical feedback of my performance as a student and as an engineer.
- My division leaders Eli Niewood and Bob Davis, for supporting my educational and professional development over the years.
- My mentor Andrew Stimac, for offering thoughtful reviews of my publications and this thesis.

Most importantly, my work would be meaningless without the love from my family. I would like to thank my parents, Kathleen and John, and my sister, Dawn, for their optimism and enthusiasm. Lastly, to my beautiful wife Bina, whose unconditional love and support continuously inspires me in all that I do.



# Contents

<b>1</b>	<b>Introduction</b>	<b>17</b>
1.1	Contributions to Robotics . . . . .	21
1.2	Technological Challenges . . . . .	22
1.3	Thesis Organization . . . . .	23
<b>2</b>	<b>Related Work</b>	<b>25</b>
2.1	Quad-Rotor Flying Robots . . . . .	26
2.2	Mobile Ad Hoc Networks . . . . .	27
2.3	Coverage Controllers . . . . .	28
<b>3</b>	<b>Robot System</b>	<b>29</b>
3.1	Quad-Rotor Platform . . . . .	29
3.1.1	Dynamics Model . . . . .	30
3.1.2	Control for Indoor Operation . . . . .	32
3.1.3	Control for Outdoor Operation . . . . .	37
3.2	Embedded Control Module . . . . .	37
3.2.1	Interface to AutoPilot Module . . . . .	38
3.2.2	Interface to Radio Module . . . . .	40
3.3	Radio Module . . . . .	40
<b>4</b>	<b>Mobile Ad Hoc Network</b>	<b>43</b>
4.1	Importance of Broadcasting State Estimates . . . . .	44
4.1.1	Broadcast Scheme . . . . .	45

4.1.2	Importance Function . . . . .	47
4.2	Location-Based Algorithm for Broadcasts . . . . .	47
4.2.1	Probabilistic Approach . . . . .	48
4.2.2	Deterministically Selecting Estimates . . . . .	48
4.3	Network Simulations . . . . .	50
4.3.1	Monte Carlo Simulations . . . . .	50
4.3.2	Voronoi Simulations on the Embedded Control Module . . . . .	52
<b>5</b>	<b>Optimal Coverage with Downward Facing Cameras</b>	<b>55</b>
5.1	Optimal Camera Placement . . . . .	55
5.1.1	Single Camera . . . . .	56
5.1.2	Multiple Cameras . . . . .	58
5.2	Distributed Control . . . . .	60
5.2.1	Control Law . . . . .	61
5.2.2	Rectangular Field of View . . . . .	61
5.2.3	Discrete Controllers . . . . .	63
<b>6</b>	<b>Robot Experiments</b>	<b>67</b>
6.1	Optimal Coverage of an Indoor Environment . . . . .	68
6.2	Network Coverage Using Simple Flooding . . . . .	70
6.3	Location-Based Algorithm . . . . .	71
6.4	Optimal Coverage of an Outdoor Environment . . . . .	73
<b>7</b>	<b>Conclusions</b>	<b>77</b>
7.1	Lessons Learned . . . . .	77
7.2	Future Work . . . . .	78
<b>A</b>	<b>Communication Protocol</b>	<b>81</b>
<b>B</b>	<b>Electrical Schematics</b>	<b>89</b>
<b>C</b>	<b>Source Code</b>	<b>93</b>

# List of Figures

1-1	This three quad-rotor flying robot system successfully demonstrated our distributed algorithm for optimal coverage with downward facing cameras [52]. Such robot teams will augment current surveillance systems to provide currently unachievable performance. . . . .	18
1-2	The same system from Figure 1-1 was deployed outdoors with off-the-shelf miniature camcorders. By creating a video mosaic, a higher resolution view is obtained of the environment compared to using a single camera. . . . .	19
1-3	A geological survey using a single quad-rotor flying robot was conducted in March 2008 by Jan Stumpf and Daniel Gurdan of Ascending Technologies GmbH (AscTec). Here we see the tree canopy on the island of Moorea, French Polynesia. . . . .	20
1-4	An example mobile ad hoc network graph from a quad-rotor flying robot experiment is plotted in Google Earth. This fully connected network consists of five flying and four stationary robots equipped with our embedded controller. . . . .	21
3-1	This photograph, courtesy of CSAIL photographer Jason Dorfman, shows one of our AscTec Hummingbird robots autonomously flying indoors using a motion capture system. . . . .	30

3-2	The dynamics model shown of a quad-rotor flying robot is used throughout this thesis. The local dynamics of the robot are described using the body frame $(x^b, y^b, z^b, \phi^b, \theta^b, \psi^b)$ , while the global orientation is described using the global frame $(x, y, z, r^x, r^y, r^z)$ . . . . .	31
3-3	The basic PD control structure of the AutoPilot control loops for pitch, roll, and yaw stabilization [24]. . . . .	33
3-4	A motion capture system measures the position of the robots and broadcasts this information wirelessly to the robot team. . . . .	34
3-5	The classic PID control structure of the embedded waypoint position control for $z_i$ and $\psi_i$ . All computations outside the motion capture system are done onboard the robot. . . . .	35
3-6	The cascaded PID control structure of the embedded waypoint position control for $x_i$ and $y_i$ . All computations outside the motion capture system are done onboard the robot. . . . .	36
3-7	The embedded control module shown has adopted the Digi XBee footprint such that the original AutoPilot connector module can be used for power and serial communication. . . . .	38
3-8	An abbreviated diagram of the NXP LPC2148 microcontroller [45]. System blocks not relevant to the operation of the embedded control module are hidden for clarity purposes. . . . .	39
3-9	The Digi XBee-PRO and Digi XBee-XSC radio modules are used for indoor and outdoor operation, respectively. . . . .	41
4-1	A simple example where robots $i$ and $j$ share a Voronoi boundary but cannot communicate their state estimates directly. This problem is easily resolved using a mobile ad-hoc network topology to route information through robot $k$ . . . . .	45

4-2	This figure shows the average propagation time for the location-based algorithm running on a $10 \times 10$ stationary robot grid. Averages were taken over 1000 Monte Carlo simulations. For small subgraphs (i.e. $2 \times 2$ ), update rates of state estimates increased with decreasing time slot lengths. For larger subgraphs, the optimal length was around $m = 7$ . . . . .	51
4-3	A testbed for simulating distributed algorithms using the embedded control module and XBee XSC. Each module locally ran a Voronoi coverage controller and wirelessly communicated state information to other modules in the virtual robot team. A communication range was implemented to add complexity to the wireless network. . . . .	52
4-4	Coverage costs are shown for a nine robot system simulated on the embedded control module running a Voronoi coverage controller. The system has a frame rate of 1.7 Hz when using a no-hop scheme ( $l = 1$ ). The system initially performs well, but its inability to multi-hop state estimates resulted in a suboptimal final configuration that does not minimize the Voronoi cost function. A simple flooding scheme ( $l = 9$ ) improved steady state performance, however, the slow frame rate of 0.2 Hz caused the system to initially oscillate in a high cost configuration. The location-based algorithm with a time slot of length $l = 3$ performed the best overall by combining fast update rates with multi-hop capabilities. The final Voronoi configurations for the algorithm and no-hop simulations are also shown. . . . .	53
5-1	An example importance profile is shown for this environment. A larger $\Phi$ implies that a particular area is of more importance. This figure also illustrates how changes in surface elevations can be represented even though the environment is in $\mathbb{R}^2$ . . . . .	57
5-2	This figure shows optical and geometric parameters used in the coverage algorithm. For now, we are assuming a circular field of view. . . .	58

5-3	An environment can have overlapping fields of view from two or more cameras. The relevant quantities involved in characterizing this scenario are defined. . . . .	59
5-4	The geometry of a camera with a rectangular field of view is shown in this figure. . . . .	62
6-1	Frame shots from an experiment with three AscTec Hummingbird quad-rotor robots are shown. After launching from the ground (Figure 6-1(a)), the three robots stabilize in an optimal configuration (Figure 6-1(b)). Then one robot is manually removed to simulate a failure, and the remaining two move to a new optimal position (Figure 6-1(c)). Finally a second robot is removed and the last one stabilizes at an optimal position (Figure 6-1(d)). The robots move so that their fields of view (which cannot be seen in the snapshots) cover the environment, represented by the white polygon. . . . .	69
6-2	The cost function during the three stages of the experiment, averaged over 19 successful experiments, is shown. The error bars denote one standard deviation. The experiments demonstrate the performance of the algorithm, and its ability to adapt to unforeseen robot failures. . .	70
6-3	An example mobile ad hoc network graph from the quad-rotor flying robot experiment is plotted over an areal image. For a system of three robots, the simple flooding scheme was successful in routing state estimates back to the base station. . . . .	71
6-4	This figure shows acquired physical data from the experiment. Robot state estimates, as recorded by the base station, included onboard temperature. Estimates transferred directly to the base station from the originating robot are labeled with $\times$ 's, while ones needing multiple hops are labeled with $\circ$ 's. . . . .	72



6-5	An example mobile ad hoc network graph from the quad-rotor flying robot experiment is plotted in Google Earth. For this nine robot system, the location-based algorithm routes state estimates through the entire team. The bounded environment from the downward facing camera coverage problem is also shown. . . . .	73
6-6	This plot shows the time stamp of the most current state estimates received by the stationary robot beta. Estimates of closer, more important robots are updated more frequently and tend to be more current, which validates the location-based algorithm. . . . .	74
6-7	A simplified plot of Figure 6-6 is shown. From the frequency of the updates, we can conclude that gamma started the closest to beta but then moved away during the experiment. Once lambda moved into communication range with beta, state estimates of the team began to be routed through lambda. . . . .	74
6-8	The embedded control modules are used to cover an outdoor environment. Off-the-shelf miniature camcorders recorded the field of view of each robot during the experiment. . . . .	75
6-9	The cost function during the three stages of the outdoor experiment. Like the indoor experiment, we demonstrated the performance of the algorithm and its ability to adapt to unforeseen robot failures. . . . .	76



# List of Tables

A.1	The state packet is sent between robots to update their state estimate lists. The first entry in the checksum loop contains the sender's estimate, which is assumed to be current and is used to synchronize teammates' clocks. . . . .	82
A.2	The global packet provides position information for target robots during indoor operation. This information overrides all other methods of position acquisition. This packet has no effect during outdoor operation.	83
A.3	The communication packet initializes TDMA communication among the robots in the team. The packet also determines which robots are in the team for high level control. Once sent, following packets must be sent in an assigned slot, even if originating from a base station. . .	84
A.4	The environment packet builds the environment boundary polygon prior to high level control initialization. The importance function over the environment is assumed to be known by the robots, otherwise importance is assumed to be unity. . . . .	85
A.5	The algorithm packet initializes high level control for target robots. Team members are determined from a prior communication packet, otherwise robots assume there is no team. . . . .	86
A.6	The dynamics packet initializes low level control for target robots. Current waypoint is set to first valid position acquired after packet is received.	87



# Chapter 1

## Introduction

The payload for an Unmanned Aerial Vehicle (UAV)<sup>1</sup> is typically considered a separate subsystem from the vehicle itself. Engineers spend much of their resources designing the payload in an effort to maximize its performance. This type of prioritizing can cause the UAV to become an afterthought; the engineers may be tempted to select a preexisting vehicle based on convenience, cost, or familiarity. Ironically, the vehicle can have as much of an impact on system performance as the payload itself. Consider a surveillance application where the payload is a downward facing camera. Increasing the altitude of the camera results in a wider field of view, much like the effect of a zoom lens. In fact, one can argue this maneuver is more effective than “zooming” since it does not change the lens speed, allowing the optics to perform identically at all vehicle configurations. Here the integration of the UAV into the payload forms the basis of a high performance robot.

This thesis concentrates specifically on the quad-rotor flying robot, an element of the rotary wing subclass of UAVs. A collaborating quad-rotor robot team, like the one seen in Figure 1-1, is capable of outperforming the traditional single (usually fixed-wing) UAV by providing multiple payload configuration capabilities. In the surveillance example, the resulting image may need to fully cover a defined environment. With a single UAV, there are limited number of configurations to accomplish

---

<sup>1</sup>Currently referred to as Unmanned Aircraft Systems (UASs) by the U.S. Department of Defense, who has modified the previous definition of UAVs to provide a working definition of an “unmanned system” [14].

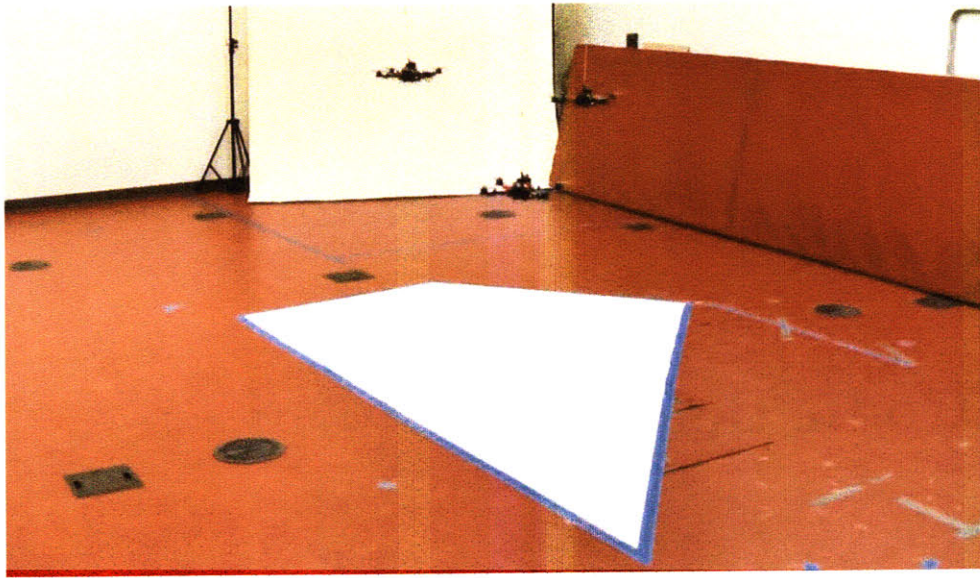


Figure 1-1: This three quad-rotor flying robot system successfully demonstrated our distributed algorithm for optimal coverage with downward facing cameras [52]. Such robot teams will augment current surveillance systems to provide currently unachievable performance.

this task, bounding the overall performance of the system. By having multiple quad-rotor flying robots, we are able to continuously place the cameras in an optimal configuration, even in a dynamic environment. This ability again allows for “zooming,” except here we are decreasing the altitude to increase resolution over areas of importance.

The use of multiple payloads also reduces the required performance specifications of the payload itself. An expensive, high-performance payload can be replaced by affordable, commercially available components. In the surveillance example, scientific grade CCDs are often fabricated to produce a high resolution, wide angle camera. Equal or superior performance is achieved by deploying several consumer cameras that are able to be positioned closer to the environment. As a result, total system cost is significantly reduced. Figure 1-2 shows one frame of a video mosaic from an outdoor coverage experiment with three quad-rotor flying robots; we were able to deploy inexpensive video cameras to obtain a high resolution view of the environment.

Such multi-robot systems are inherently robust against robot failures. If failures

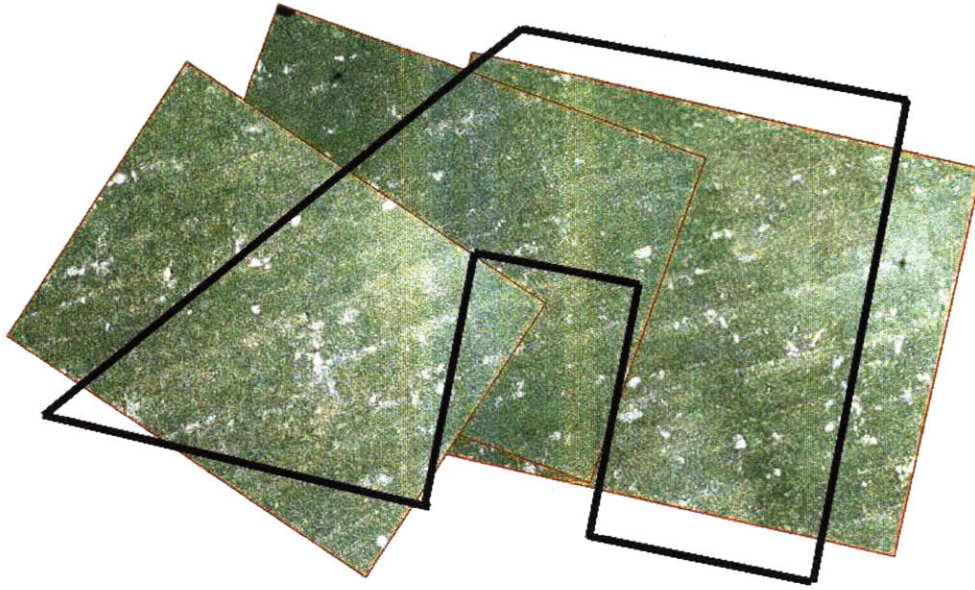


Figure 1-2: The same system from Figure 1-1 was deployed outdoors with off-the-shelf miniature camcorders. By creating a video mosaic, a higher resolution view is obtained of the environment compared to using a single camera.

occur during operation, the system can accommodate for the lost payloads by reconfiguring itself. This situation is catastrophic for a single UAV system; vehicle failure often results in mission failure. By making the system purely decentralized, we can completely avoid dependencies on a central controller, further increasing robustness. In addition, such autonomy allows the team to accomplish missions that are normally considered “out of range” of the ground stations [50]. Thus, we are not only motivated to design an onboard embedded controller for the quad-rotor flying robots, but have it be capable of running algorithms in distributed fashion.

Applications for a distributed quad-rotor robot team are not limited to surveillance. Figure 1-3 shows a geological survey mosaic produced from multiple images taken by one quad-rotor flying robot. Using multiple robots would provide wider tree canopy coverage over the same period of time. They could also be used in a similar fashion to provide valuable information to firefighters during a forest fire. In fact, collaborating quad-rotor flying robots can be used to assist rescuers in many natural disaster situations: deploying motion and acoustic sensors to locate earthquake



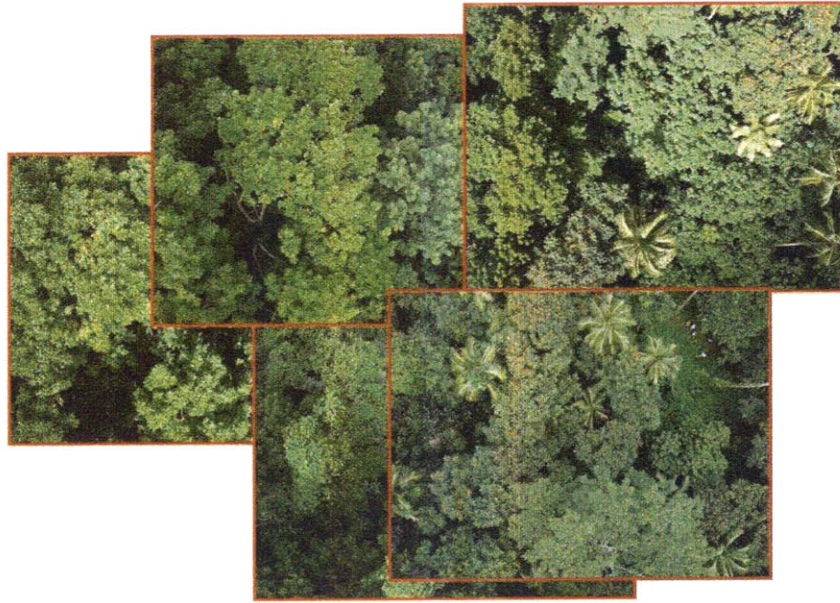


Figure 1-3: A geological survey using a single quad-rotor flying robot was conducted in March 2008 by Jan Stumpf and Daniel Gurdan of Ascending Technologies GmbH (AscTec). Here we see the tree canopy on the island of Moorea, French Polynesia.

victims; distributing food and first aid supplies to stranded flood victims; and monitoring lava flow to maximize time for evacuation during volcanic events. The defense applications are just as plentiful; these systems are expected to perform many of the dull, dirty, and dangerous roles currently performed by the warfighter [14].

We believe research in distributed quad-rotor robotics will have its greatest impact in telecommunications. The ability to act as wireless routers allows these robots to form a three dimensional mobile ad hoc network, like the one in Figure 1-4. When combined with advanced localization techniques [28], this network can extend into urban environments denied of a reliable GPS signal. The ultimate vision of the DoD is for such an aerial robot team to interact with unmanned ground and land systems to extend and augment the capabilities of the warfighter [14].



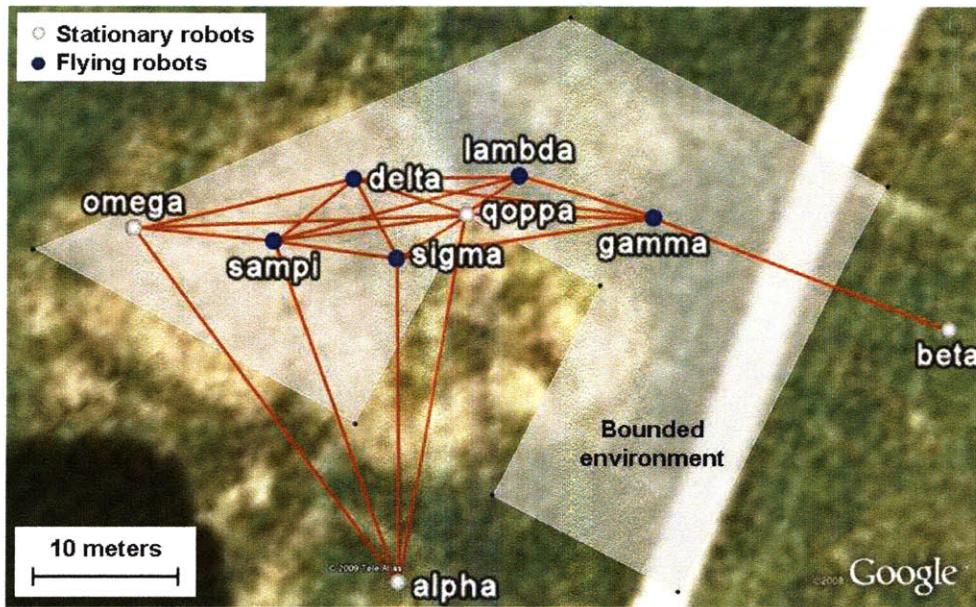


Figure 1-4: An example mobile ad hoc network graph from a quad-rotor flying robot experiment is plotted in Google Earth. This fully connected network consists of five flying and four stationary robots equipped with our embedded controller.

## 1.1 Contributions to Robotics

For our system of quad-rotor flying robots, we designed an onboard embedded controller capable of running algorithms in distributed fashion. For indoor operation, we use a motion capture system to wirelessly relay position information for low level control. For outdoor operation, a commercially available module allows for GPS, altitude, and compass waypoint control. In addition, we implement a mobile ad hoc network infrastructure to propagate state estimates necessary for the robots to self-organize. This system provides us with a high performance testbed for implementing distributed algorithms designed for a networked team of hovering robots.

In this thesis, we provide the following contributions to the field of robotics:

1. *Embedded Control Module Design* - Hardware and software designs are discussed for the embedded control of a quad-rotor flying robot. The control structure for both indoor and outdoor flight is presented with parameters specific to a commercially available robot platform.

2. *Information Management for Distributed Algorithms* - Embedded software routines are constructed to provide the distributed algorithm with real-time system information. Low level, fast executing functions supporting overall system stability are given priority over high level, slow executing processes.
3. *Location-Based Multi-Hop Strategy* - A deterministic algorithm is presented that efficiently propagates state estimates among a robot team without the need of a routing scheme. The algorithm is shown to be effective in solving distributed control problems while being efficient in terms of communication bandwidth and computational complexity.
4. *Optimal Coverage with Downward Facing Cameras* - The embedded control module is used to optimally position downward facing cameras installed on the quad-rotor flying robots over a bounded environment. Experiments are conducted using both the indoor and outdoor flight configurations.

## 1.2 Technological Challenges

There were numerous hardware, software, and control challenges we faced during development. The advanced dynamics of the quad-rotor robot inherently makes the control system more complicated than traditional fixed-wing UAVs; the fast time-response of the robot plant that allowed for indoor flight is the same characteristic that sometimes led to unstable behavior. The robots also experienced dynamic coupling between the high level distributed algorithm and low level control, which required careful tuning of feedback parameters and control loop rates. We understand that the final configuration may not be theoretically ideal, but instead results in acceptable performance in the heuristic sense.

There were also many challenges associated with acquiring state estimates. Our urban location at the Massachusetts Institute of Technology (MIT) often resulted in an inaccurate GPS and/or magnetic compass readings. In addition, the pressure sensors used for altitude control experience significant drifts with time, a characteristic

that had to be accounted for during experiments. These difficulties with outdoor flight made using the indoor motion capture system appealing, although the limited sensing volume restricted us to a maximum of three simultaneous flying robots.

Once state estimates are acquired, they need to be communicated among the robot team. Here we experienced inconsistencies in connectivity and performance of the radio technology, especially in noisy environments. We do not yet understand how to construct an accurate communication model, which prevents us from considering all possible modes of failure. The distributed algorithms we develop are robust against communication failure, but the system supervisor functions (e.g. collision detection) are not.

### **1.3 Thesis Organization**

This thesis is divided into seven chapters. Chapter 2 presents related work on quadrotor flying robots, mobile ad hoc networks, and coverage algorithms. Chapter 3 discusses the robot platform and the embedded controller hardware for decentralized control. Chapter 4 introduces a novel location-based algorithm for multi-hopping state estimates with the robot team. Chapter 5 presents a novel distributed algorithm that moves the robot team with downward facing cameras to optimally cover a bounded environment. Chapter 6 discusses the experiments from the system and algorithms we describe in the previous chapters. We conclude in Chapter 7 with final thoughts and lessons learned.



# Chapter 2

## Related Work

Since Archibald Low's invention of the Aerial Target in 1916 [57], UAVs have been one of the most popular research topics in engineering. By removing the human from the aircraft, a vehicle of a smaller size (and thus smaller signature) can perform assignments with increased survivability, endurance, and maneuvering capabilities [14]. Rotary wing UAVs have distinct advantages over conventional fixed-wing aircrafts since they can takeoff and land in limited spaces and easily perform stationary hovers [6]. Compared with a conventional helicopter with a single large rotor, a quad-rotor flying robot uses much smaller rotors that allow for closer approaches to obstacles without fear of rotor strike [33, 47].

By extending the work of general UAV teams, we believe the distributed quad-rotor platform can significantly increase the performance of current mobile sensor networks in tasks such as surveillance [15], target tracking [10, 13, 39], and team navigating [49]. We are inspired by numerous research platforms designed for the autonomous operation of multiple UAVs. How et al. at MIT have developed a system capable of evaluating coordination and control algorithms for teams of eight fixed-wing autonomous UAVs [36]. Beard et al. at Brigham Young University have constructed an experimental platform for decentralized path following control of small air vehicles in high-wind conditions [9]. This list is far from comprehensive, as research labs at Stanford University [58], University of California Berkeley [54], Vanderbilt University [21], and University of Essex [34] have also developed such systems.

## 2.1 Quad-Rotor Flying Robots

To date, a significant amount of research on quad-rotor flying robots has focused on the dynamics of a single vehicle. Early work by Hauser et al. studied dynamic models for nonlinear control of a vertical takeoff and landing (VTOL) aircraft [27], from which Shim et al. compared several control designs for autonomous helicopters [55]. In 2002, Pounds et al. and Hamel et al., whom both included Robert Mahony, published dynamic models for the quad-rotor flying robot known as the X4-flyer [26, 48]. The X4-flyer was further modified to produce favorable stability properties in [47]. Recent work in characterizing the flight dynamics of a quad-rotor flying robot has led to several variations of inertial-based controllers [19, 33, 43]. In particular, the high frequency controller proposed by Gurdan et al. will be used for this thesis [24].

A popular extension of autonomous quad-rotor flight is through the use of vision-based control [23]. Altug et al. used a ground-based, two camera pose estimation method to control a quad-rotor flying robot [7]. Soon after Tournier et al. implemented a fixed moiré target to sustain a stable hover with a robot mounted camera [59], while Kemp employed a dynamic programming algorithm to develop a filter capable of vision-based control [38]. Flowers et al. furthered this work by developing an onboard FPGA platform to minimize drift via Harris feature detection and template matching [20].

Other researchers have concentrated on autonomous trajectory tracking and motion planning. Salazar-Cruz et al. implemented a nested saturation control algorithm to track a global trajectory [51]. Bouktir et al. proposed a minimum time trajectory method derived from a nonlinear optimization using algebraic spline functions [11], while Meister et al. characterized numerous collision avoidance strategies based on range sensing capabilities [42]. He et al. navigated a quad-rotor flying robot indoors using an Unscented Kalman Filter extension of the Belief Roadmap algorithm [28]. This work under Nicholas Roy at MIT was the basis of further research efforts to investigate simultaneous localization and mapping (SLAM) using stereo vision and laser odometry [4, 5, 8]

Concerning multi-agent systems, the literature has been somewhat limited for quad-rotor implementation. The Real-time indoor Autonomous Vehicle test Environment (RAVEN) developed in the Aerospace Controls Laboratory at MIT studies long-duration multivehicle missions in a controlled environment [35]. The Stanford Testbed of Autonomous Rotorcraft for Multi-Agent Control (STARMAC) developed in the Hybrid Systems Laboratory at Stanford University uses quad-rotor flying robots to autonomously track a given waypoint trajectory [30]. At the time of our work, both RAVEN and STARMAC relied on a centralized computer for multi-agent coordination. Recent work to STARMAC is pursuing onboard optimal control strategies [3], for which Hoffmann et al. have published advances in decentralized collision avoidance [31] and information-seeking guidance [32].

## 2.2 Mobile Ad Hoc Networks

Concerning our algorithm for mobile ad hoc networks, a substantial body of work exists on location-based routing. Haas proposed a zone-based routing protocol using a radius parameter to reduce the number of control messages [25]. Ni et al. developed a distance-based scheme to decide when a node should drop a rebroadcast [44], while Sun et al. adapted a similar scheme for setting defer times [56]. Ying et al. discussed how these ad hoc schemes influence flooding costs [17].

Our proposed algorithm is related to this body of work in that location is used to broadcast information through a mobile ad hoc network. However, instead of routing actual data packets to a predetermined receiver, we are deterministically transmitting state information to be used by the entire team of robots. This allows all transmissions to be treated as simple broadcasts, for which the sender uses the algorithm to select state estimates. This strategy is applicable for many distributed control problems, such as coverage control algorithms for mobile sensing networks.

## 2.3 Coverage Controllers

For our work on optimal coverage with downward facing cameras, we are inspired by recent literature concerning the optimal deployment of robots for providing sensor coverage of an environment. Cortes et al. introduced a stable distributed controller for sensor coverage [16] based on ideas from the optimal facility placement literature [18]. This approach involves a Voronoi partition of the environment, and has experienced several extensions [22, 46, 53]. One recent extension described in [41] proposed an algorithm for the placement of hovering sensors, similar to our scenario.

Our proposed method is related to this body of work in that the distributed control algorithm is derived by taking the gradient of a cost function. However, the cost function we propose does not involve a Voronoi partition. Instead, it relies on the fields of view of multiple cameras to overlap with one another. Another distinction from previous works is that the configuration space of the robots is different from the coverage space. This allows for optimal coverage of complex (non-convex, disconnected) environments [52].



# Chapter 3

## Robot System

The quad-rotor flying robot fleet at DRL<sup>1</sup> is composed of five Ascending Technologies (AscTec) Hummingbirds [2]. Each is equipped with an AscTec AutoPilot module to stabilize the robot in pitch, roll, and yaw. In addition, an onboard GPS module, pressure sensor, and magnetic compass allows for GPS, altitude, and compass waypoint control during outdoor operation. We developed an onboard embedded control module that acquires state estimates from the robot and wirelessly communicates them to other robots using commercially available radio modules. The same microcontroller runs the distributed control algorithm, which allows the robot team to self-organize in both indoor (motion capture system) and outdoor (GPS enabled) environments.

### 3.1 Quad-Rotor Platform

The AscTec Hummingbird, as shown in Figure 3-1, is a classic four-rotor helicopter with clockwise and counterclockwise rotating propellers located on the forward and side axes, respectively<sup>2</sup>. All propellers are equidistant from the center of the vehicle, providing an under-actuated platform ideal for hovering.

Consider steady-state operation when all four propellers are producing equal thrust totaling the force of gravity on the vehicle. Since the pair of counter rotating

---

<sup>1</sup>The Distributed Robotics Laboratory at MIT.

<sup>2</sup>The clockwise/counterclockwise pair ordering is irrelevant; the clockwise and counterclockwise rotating propellers could have been located on the side and forward axes, respectively.

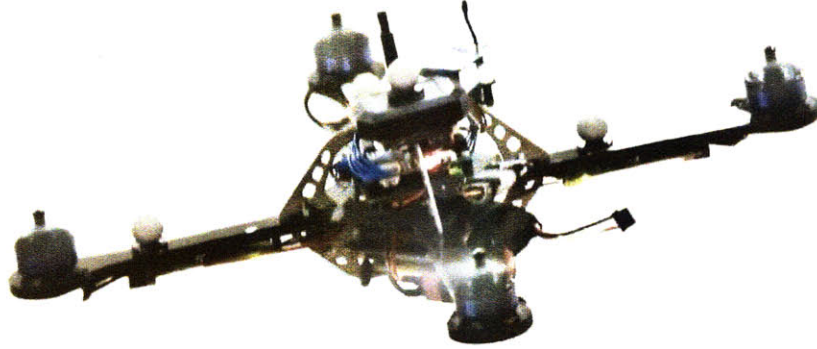


Figure 3-1: This photograph, courtesy of CSAIL photographer Jason Dorfman, shows one of our AscTec Hummingbird robots autonomously flying indoors using a motion capture system.

propellers are spinning at equal speeds, the moment of inertia about the downward-axis is balanced, resulting in a stable hover. Movement about the downward axis is accomplished by increasing or decreasing thrust equally on all propellers. To provide translational forward movement, a differential thrust is produced between the clockwise rotating propellers to tip the robot about the side-axis. Due to propeller symmetry, all gyroscopic effects are balanced, resulting in a stable pitch. Translational side movement is similarly produced using the counterclockwise rotating propellers to roll about the forward-axis. For rotational movement about the downward-axis, a differential thrust is produced between the pair of clockwise and counterclockwise rotating propellers. As long as propellers of like rotation remain at equal speeds and total thrust balances the gravitational force, a stable yaw results.

### 3.1.1 Dynamic Model

The dynamic model for a quad-rotor flying robot is shown in Figure 3-2. The body frame defines the robot's translational vectors of forward, right, and downward as positive  $x^b$ ,  $y^b$ , and  $z^b$ , respectively. The Euler rotational angles of roll, pitch, and

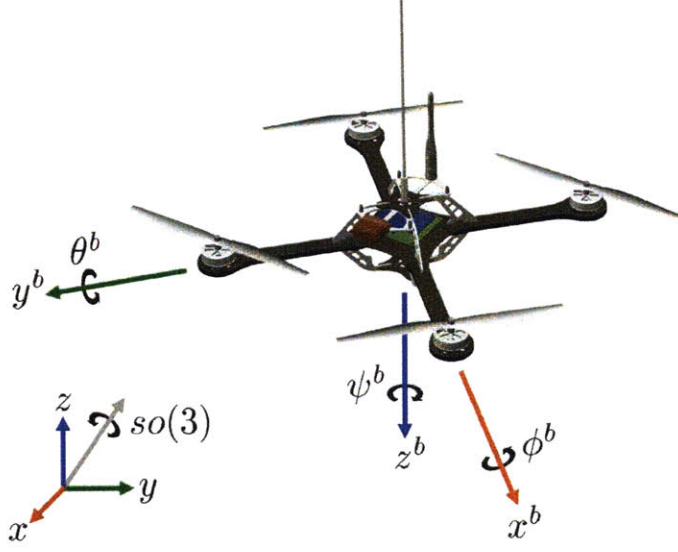


Figure 3-2: The dynamic model shown for a quad-rotor flying robot is used throughout this thesis. The local dynamics of the robot are described using the body frame  $(x^b, y^b, z^b, \phi^b, \theta^b, \psi^b)$ , while the global orientation is described using the global frame  $(x, y, z, r^x, r^y, r^z)$ .

yaw are defined as positive  $\phi^b$ ,  $\theta^b$ , and  $\psi^b$ , respectively. When describing the local dynamics of a given robot  $i$ , the body state is defined as  $\vec{p}_i^b = [x_i^b, y_i^b, z_i^b, \phi_i^b, \theta_i^b, \psi_i^b]^T$  in  $\mathbb{R}^3 \times \mathfrak{S}^3$ . However, the global state  $\vec{p}_i = [x_i, y_i, z_i, r_i^x, r_i^y, r_i^z]^T$  in  $\mathbb{R}^3 \times \mathfrak{S}^3$  is used to locate the robot with respect to an earth-based environment, where  $r^x$ ,  $r^y$ , and  $r^z$  are the  $so(3)$  angles of the global axes  $x$ ,  $y$ , and  $z$ .

For transformations of translational vectors from the global frame to the body frame, we define the rotational matrix

$$\mathbf{R}_b(\vec{p}_i) = \begin{bmatrix} t_c r_i^y r_i^x + r_i^z \sin(\Psi_i) & t_c r_i^y r_i^y + \cos(\Psi_i) & t_c r_i^y r_i^z - r_i^x \sin(\Psi_i) \\ t_c r_i^x r_i^x + \cos(\Psi_i) & t_c r_i^x r_i^y - r_i^z \sin(\Psi_i) & t_c r_i^x r_i^z + r_i^y \sin(\Psi_i) \\ -t_c r_i^z r_i^x + r_i^y \sin(\Psi_i) & -t_c r_i^z r_i^y - r_i^x \sin(\Psi_i) & -t_c r_i^z r_i^z - \cos(\Psi_i) \end{bmatrix} \quad (3.1)$$

where  $\Psi_i = \sqrt{r_i^x r_i^x + r_i^y r_i^y + r_i^z r_i^z}$  and  $t_c = 1 - \cos(\Psi_i)$ . Here,  $\mathbf{R}_b(\vec{p}_i)\vec{q}$  transforms the vector  $\vec{q} \in \mathbb{R}^3$  from the global frame to the body frame of robot  $i$ . In addition, we

define the yaw of the robot to be

$$\psi_i = \arctan 2(-t_c r_i^y r_i^x - r_i^z \sin(\Psi_i), t_c r_i^x r_i^x + \cos(\Psi_i)) \quad (3.2)$$

which aligns with the  $z$ -axis of the global coordinate system.

### 3.1.2 Control for Indoor Operation

The onboard AutoPilot module uses a three-axis accelerometer and three-axis gyroscope to provide attitude stabilization for the quad-rotor flying robot. Let  $\vec{\gamma}_i^f = [\ddot{\phi}_i^f, \ddot{\theta}_i^f, \ddot{\psi}_i^f]^T$  in  $\mathbb{R}^3$  be the rotational acceleration estimates from the fused inertial data. Three independent proportional-derivative (PD) control loops use the inertial data to update the torque input to all rotors at 1 kHz. Let  $\vec{u}_i^b = [u_{\phi_i}^b, u_{\theta_i}^b, u_{\psi_i}^b, u_{z_i}^b]^T$  in  $\mathcal{U}_z^3 \times \mathcal{U}_z$  be the pitch, roll, yaw, and thrust command inputs to the AutoPilot, respectively, where  $\mathcal{U}_z$  and  $\mathcal{U}_z$  are defined in the AutoPilot technical manuals [1]. The pitch, roll, and yaw command inputs are scaled and summed with both the proportional and derivative terms to provide a “heading-lock” control structure. The thrust input is scaled and summed directly with the torque input to all rotors. This attitude controller, which is based on the work of Gurdan et. al [24], is shown in Figure 3-3.

We conducted autonomous indoor flights using a CSAIL<sup>3</sup> laboratory equipped with a Vicon motion capture system. This system uses 16 high resolution infrared (IR) cameras to triangulate the  $\mathbb{R}^3$  global coordinates of IR reflective spherical markers in the sensing environment. As seen in Figure 3-1, these markers are installed on each robot in a predetermined, unique configuration. Careful consideration is taken to avoid Euclidean symmetry among all configurations. By constructing a digital model of each configuration, the motion capture system is able to measure the global state  $\vec{p}_i$  of each robot at a rate of 120 Hz. The states are then broadcasted wirelessly at a rate of 50 Hz. The system configuration is shown in Figure 3-4.

For waypoint position control, we designed an embedded control module that runs two different proportional-integral-derivative (PID) control loop configurations. Let

---

<sup>3</sup>The Computer Science and Artificial Intelligence Laboratory at MIT.

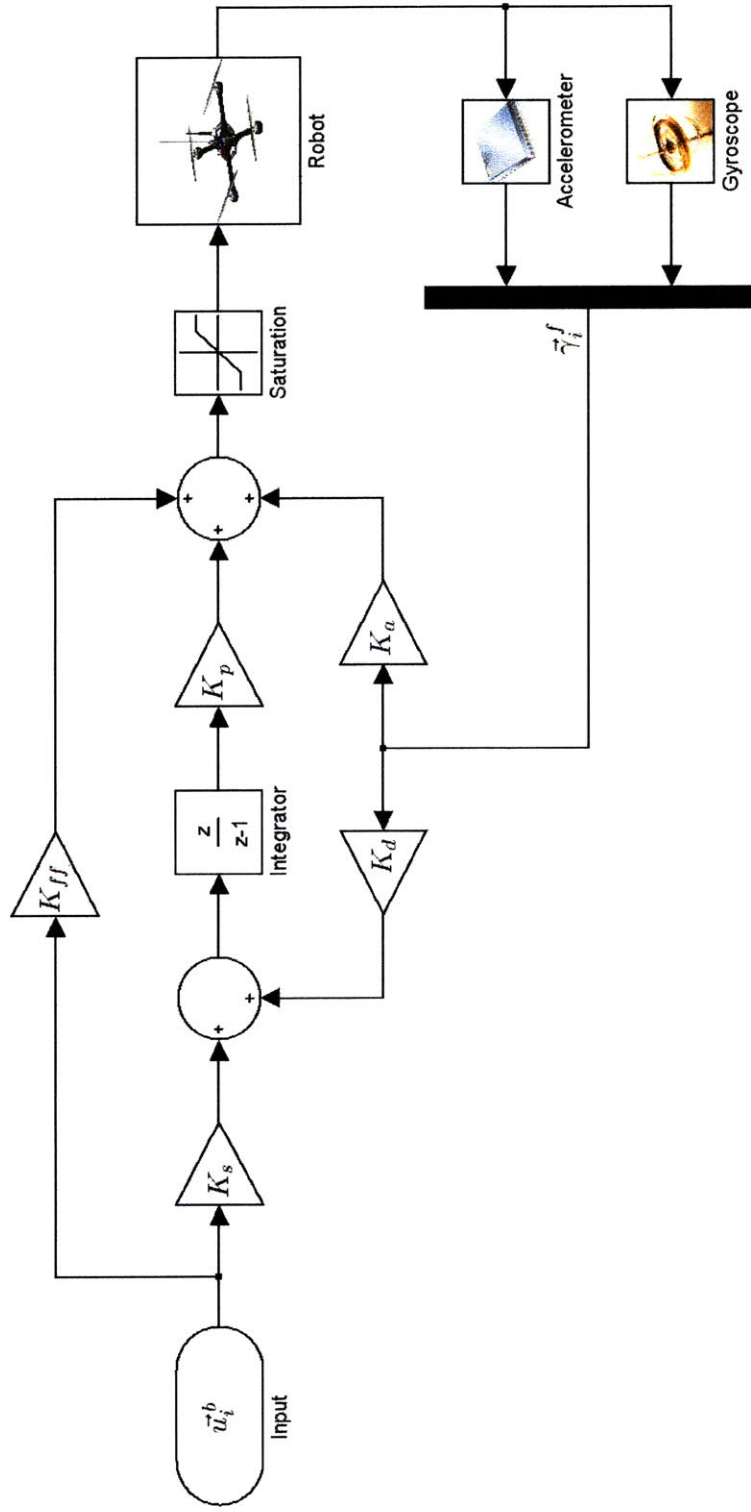


Figure 3-3: The basic PD control structure of the AutoPilot control loops for pitch, roll, and yaw stabilization [24].

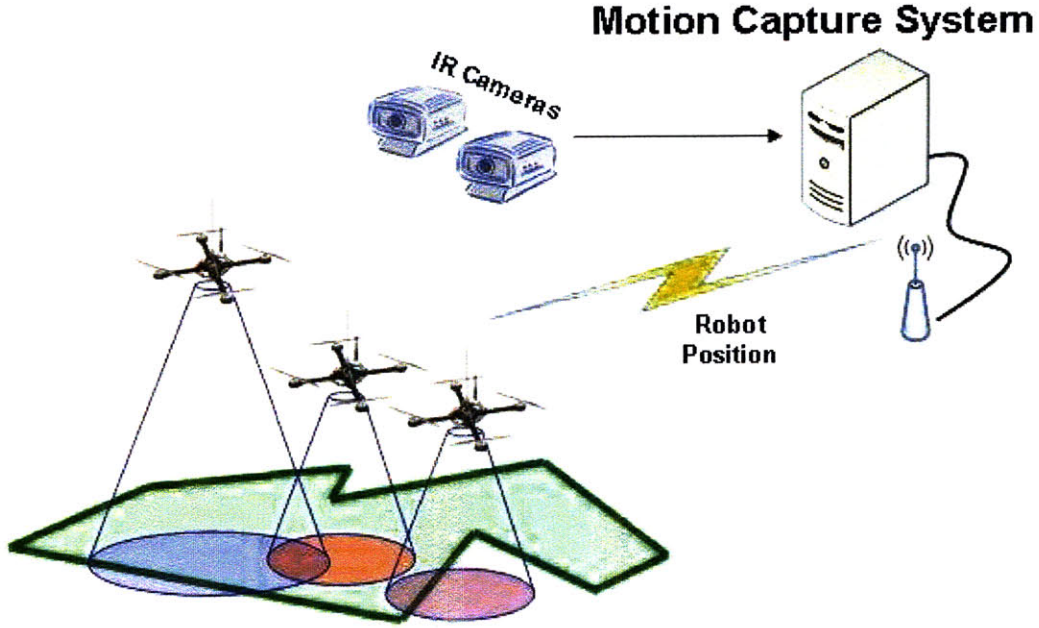


Figure 3-4: A motion capture system measures the position of the robots and broadcasts this information wirelessly to the robot team.

$\vec{w}_i = [x_w, y_w, z_w, \psi_w]^T$  in  $\mathbb{R}^3 \times \mathcal{S}$  be the current waypoint of robot  $i$  in the global frame. The embedded control module first acquires the broadcasted position data via a radio module and subtracts it from the waypoint to give the global error vector

$$\vec{e}_i = \begin{bmatrix} x_w - x_i \\ y_w - y_i \\ z_w - z_i \\ \psi_w - \psi_i \end{bmatrix} \quad (3.3)$$

where  $\psi_i$  is defined in Equation (3.2). The global error vector is then rotated to align with the body frame of robot  $i$  to give

$$\vec{e}_i^b = \begin{bmatrix} \mathbf{R}_b(\vec{p}_i) & 0 \\ 0 & 1 \end{bmatrix} \vec{e}_i \quad (3.4)$$

which feeds into two different PID modules. For yaw and thrust, the standard PID

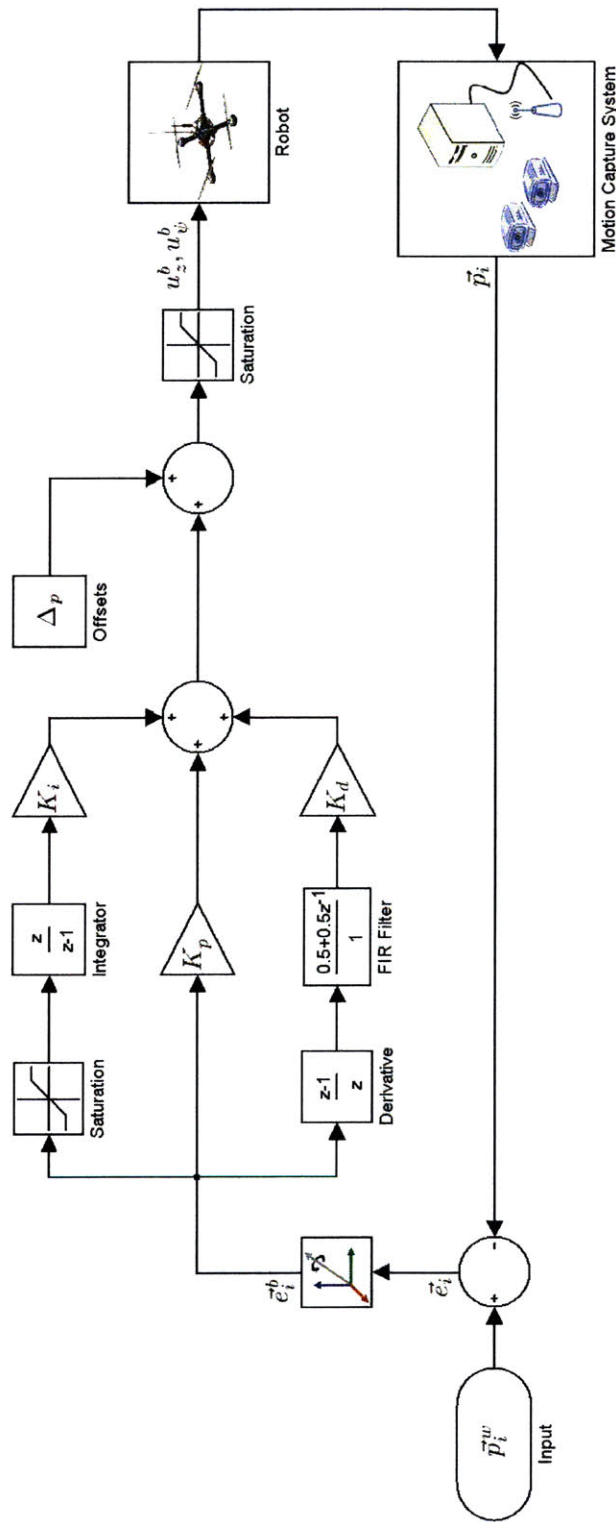


Figure 3-5: The classic PID control structure of the embedded waypoint position control for  $z_i$  and  $\psi_i$ . All computations outside the motion capture system are done onboard the robot.



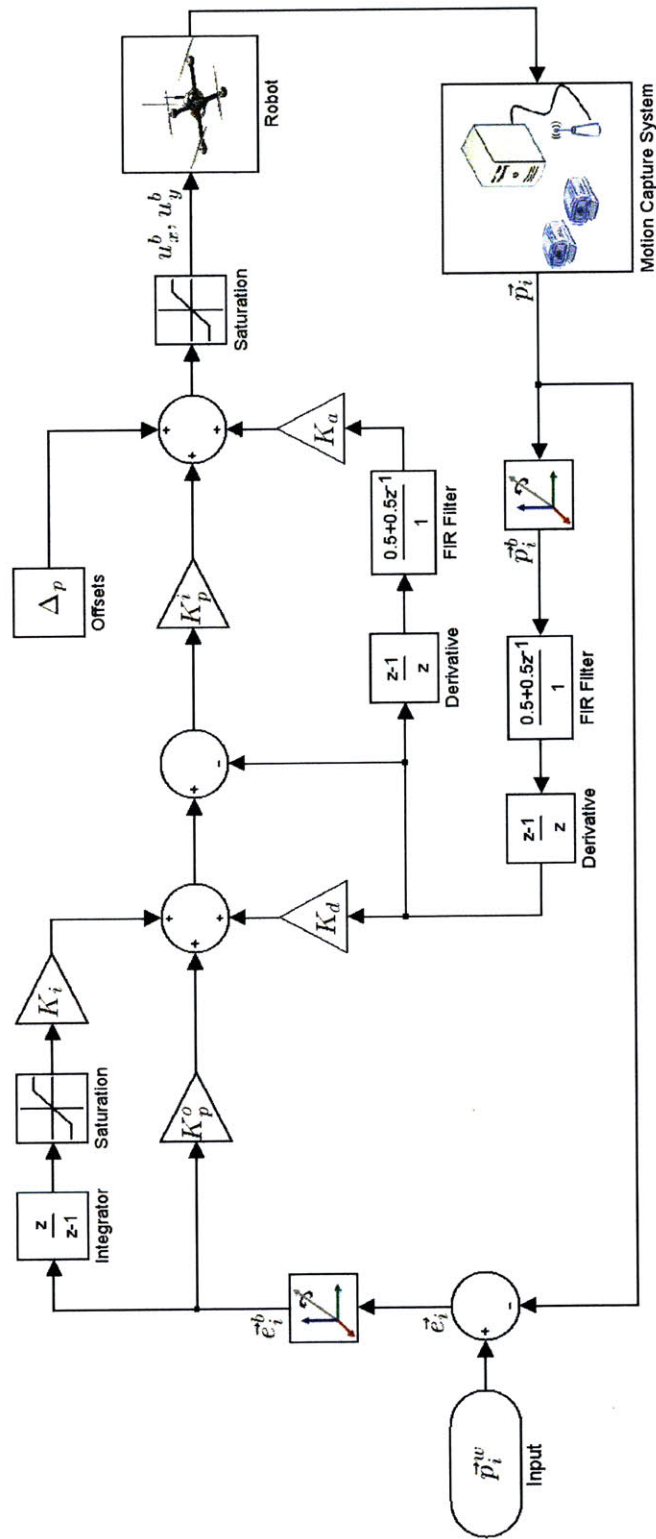


Figure 3-6: The cascaded PID control structure of the embedded waypoint position control for  $x_i$  and  $y_i$ . All computations outside the motion capture system are done onboard the robot.



control loop shown in Figure 3-5 is used since  $[u_{\psi_i}^b, u_{z_i}^b]$  is proportional (although nonlinear) to  $[\psi_i^b, z_i^b]$ . However, this relationship does not exist between  $[u_{\phi_i}^b, u_{\theta_i}^b]$  and  $[x_i^b, y_i^b]$ . Thus, the cascaded PID control loop shown in Figure 3-6 is used for pitch and roll. Both control loops incorporate low pass filters for the discrete derivative blocks, saturation limits for the discrete integrator blocks, and adjustable output offsets. This embedded waypoint position control system produces command inputs to the AutoPilot module at a rate of 33 Hz.

### 3.1.3 Control for Outdoor Operation

The AutoPilot module uses a GPS module, pressure sensor, and magnetic compass in combination with the inertial data to provide outdoor waypoint control for the quad-rotor flying robot. To use this off-the-shelf waypoint control, we programmed the embedded control module to provide global waypoint position commands,  $\vec{p}_i^w$ , to the AutoPilot module instead of lower level command inputs,  $\vec{u}_i^b$ . In addition, the state estimates,  $\vec{p}_i$ , are polled from the AutoPilot module to be used by our high level distributed algorithm. We designed the waypoint structure to be identical for both indoor and outdoor operation, which allows the high level algorithm to be agnostic to the system's low level control.

## 3.2 Embedded Control Module

The embedded control module is designed around a single NXP LPC2148 microcontroller [45]. The ARM7TDMI-S microcontroller possesses a 32-bit wide data path, 32 (+8) kB of static random access memory (SRAM), and 512 kB of flash program memory. Figure 3-8 shows a diagram of the microcontroller architecture relevant to the processes described in this thesis. We operate the microcontroller with an external crystal oscillating at 14.7456 MHz. Combined with a internal phase lock loop (PLL) multiplier of  $4x$ , we achieve a system clock frequency of 58.9824 MHz. The ARM7TDMI-S processor operates at this frequency on the local bus to control the SRAM, flash memory, and fast general purpose input/output (FIO). The vectored

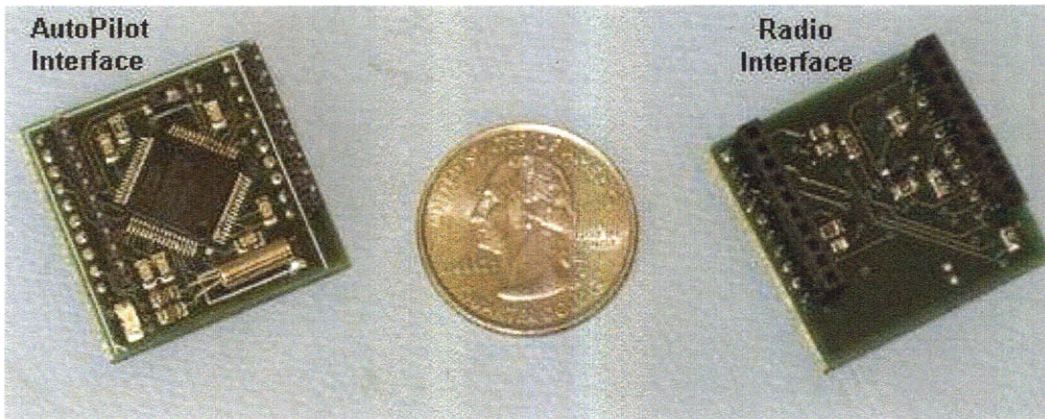


Figure 3-7: The embedded control module shown has adopted the Digi XBee footprint such that the original AutoPilot connector module can be used for power and serial communication.

interrupt controller (VIC) operates on the advanced high-performance bus (AHB) to process interrupt requests during code execution.

Bridging to the AHB is the VLSI peripheral bus (VPB), which connects all peripheral blocks at the system clock frequency implemented by a unity VPB divider. The system control block maintains several low level system features not related to specific peripheral devices (e.g. memory mapping). The two system timers (TMR0 and TMR1) are extensively used in the embedded software to trigger high priority interrupts and control external match pins for status LEDs. Two universal asynchronous receive/transmit (UART0 and UART1) are used to communicate to the AutoPilot and radio modules, respectively. Since UART0 does not support automatic flow control, an external interrupt (EINT) pin and a FIO pin are used for manual implementation. One additional EINT pin is connected to a mechanical switch to allow in-situ changes between indoor and outdoor operation.

### 3.2.1 Interface to AutoPilot Module

The embedded control module interfaces with the AutoPilot module via six electrical signals: ground (GND), 3.3V power (VD3.3), transmit output (TXD\_AP), receive input (RXD\_AP), request to send (RTS\_AP), and clear to send (CTS\_AP). As shown

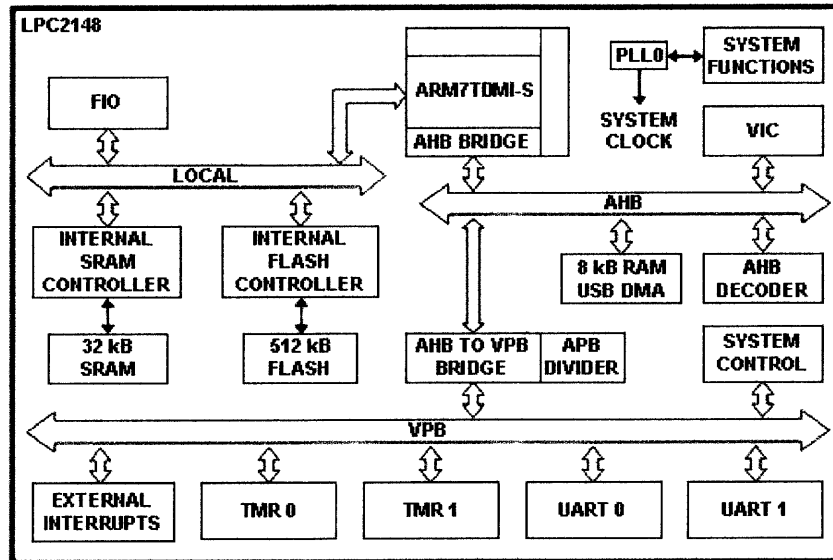


Figure 3-8: An abbreviated diagram of the NXP LPC2148 microcontroller [45]. System blocks not relevant to the operation of the embedded control module are hidden for clarity purposes.

in Figure 3-7, the socket provided by the AscTec connector board has the same footprint as the Digi XBee module in Figure 3-9. The GND signal is the floating level of the negative terminal of the robot's lithium-ion battery. The VD3.3 signal is a filtered voltage produced by a low noise linear regulator located on the AscTec connector board. We use this signal as our power source for the control and radio modules.

TXD\_AP and RXD\_AP are connected to the UART1 receive and transmit pins, respectively. This communication channel is configured for 8-bit words with no parity and 1 stop bit transmitted at 57.6 kbps. Outgoing information from UART1 is held in its 16-byte transmit buffer until the AutoPilot module pulls RTS\_AP low. This signal is connected to the clear to send pin of UART1. UART1 automatically manages flow control, sending data from the transmit buffer over RXD\_AP until all data is sent or the AutoPilot module pulls RTS\_AP high. Outgoing information from the AutoPilot module is requested by pulling the CTS\_AP low with the request to send pin of UART1. UART1 automatically pulls CTS\_AP high when the 16-byte receive buffer contains at least 8 bytes. CTS\_AP is reasserted as soon as the receive buffer

contains 4 bytes or less.

### 3.2.2 Interface to Radio Module

The control module interfaces with the radio module via eight electrical signals: GND, VD3.3, reset (RST), bootloader (BST), transmit output (TXD\_RF), receive input (RXD\_RF), request to send (RTS\_RF), and clear to send (CTS\_RF). As previously stated, power over VD3.3 is drawn from the AscTec connector board. RXD\_RF and TXD\_RF are connected to UART0 in identical fashion with respect to the AutoPilot module and UART1. However, UART0 does not support automatic flow control. Instead, RTS\_RF is connected to an EINT pin, which triggers an interrupt to manually enable or disable the data transfer from UART0. Likewise, CTS\_RF is connected to a FIO pin, whose level is set during UART0 interrupts based upon the state of the receive buffer.

RST and BST are only used when programming the microcontroller. Each signal originates from a digital output pin on the radio module. By pulling RST low while BST is low, the microcontroller enters a bootloader state, which allows UART0 to write a binary program file into the flash memory. This configuration allows us to reprogram the microcontroller wirelessly while the control module is installed on the robot.

## 3.3 Radio Module

As shown in Figure 3-9, we use two different, yet interchangeable, IEEE 802.15.4 radio modules, depending on the environment. For indoor operation, we use a Digi XBee-PRO module that operates in the 2.4 GHz frequency band. Ideally, this module can transmit at 250 kbps with a maximum line-of-sight range of 1.6 km, although in testing we have experienced performance ceilings of typically one-fourth these values. For outdoor operation, we use a Digi XBee-XSC module that operates in the 900 MHz frequency band. Like the XBee-PRO, we underperform the reported specifications of 9.6 kbps and 9.6 km.



Figure 3-9: The Digi XBee-PRO and Digi XBee-XSC radio modules are used for indoor and outdoor operation, respectively.

Although these XBee modules are capable of more complex network topologies, they are used as simple broadcast beacons to be heard by all other modules in communication range. Prior to robot deployment, all robots are programmed to use the same channel. Integrated retries and acknowledgement capabilities are disabled since network flooding schemes do not use the concept of “destination” nodes. This configuration requires manual implementation of channel access methods, which we will discuss in Chapter 4. Our goal is to reduce total communication overhead such that the baud rate is maximized.



# Chapter 4

## Mobile Ad Hoc Network

Robots in a team need to communicate state estimates to self-organize. Since many applications require the team to spread over large-scale domains, resulting distances between any two robots can become larger than their capable peer-to-peer transmission ranges. These configurations require multi-hop networking to distribute state information over the entire system. To facilitate the transportation of data packets in a multi-hop fashion, many mobile ad hoc networks implement sophisticated routing schemes. Due to the mobile nature of such networks, these schemes consume a significant amount of communication capacity for maintaining knowledge about network topology. While some routing strategies take spatial configurations into account, the robots are agnostic to the relevance of the actual data being transferred. There is no concept of data importance from the robots' point of view, often resulting in the suboptimal allocation of communication resources (e.g. time, bandwidth, power) to transfer packets.

The strategy in this chapter<sup>1</sup> allows robots to better manage communication resources for relaying state estimates. Since the collaboration of robots takes place in the physical world, spatial relationships between robot states can give insight into the importance of transferring each estimate. This location-based approach gives a quantitative answer to the question: how important is it for one robot to broadcast state information about another robot? We represent the importance of transmitting

---

<sup>1</sup>The majority of this chapter was published in [37]



a state estimate as a function that is inversely proportional to the distance between robots.

From this importance function we develop a deterministic algorithm that ensures state estimates propagate throughout a robot network. The proposed location-based algorithm is efficient in terms of bandwidth and computational complexity; it does not require network topology information to be transmitted or computed. We used Monte Carlo simulations to show increased propagation rates of state estimates in local neighborhoods. Then with real control and wireless hardware, we simulated a nine robot team running a Voronoi coverage controller to show the algorithm's effectiveness in solving distributed control problems. Experimental results for the propagation of state estimates are presented in Chapter 6 for five AscTec Hummingbird quad-rotor flying robots and four stationary robots.

## 4.1 Importance of Broadcasting State Estimates

A common assumption for distributed control algorithms is that robots have access to state estimates of other nearby robots. This assumption is often translated into unrealistic requirements on communication range. The most common requirement is that estimates need to be directly shared between robots that are within a specified distance. Another common requirement is for information to be shared between robots of a defined spatial relationship (e.g. adjacent Voronoi regions [16] or overlapping fields of view [52]).

These communication requirements are too simplistic to be realized in practice. Actual network topologies depend on more than simple distance criteria, such as environment geometry, channel interference, or atmospheric conditions. Even if transmission ranges are ideal in the physical sense (e.g. the ideal disk model), spatial relationships for certain distributed controllers cannot guarantee peer-to-peer connectivity. Figure 4-1 shows a configuration where a direct communication link cannot be created between the Voronoi neighbors  $i$  and  $j$ . Moreover, robots that are spatially disconnected may decide not to route state estimates to one another. If they



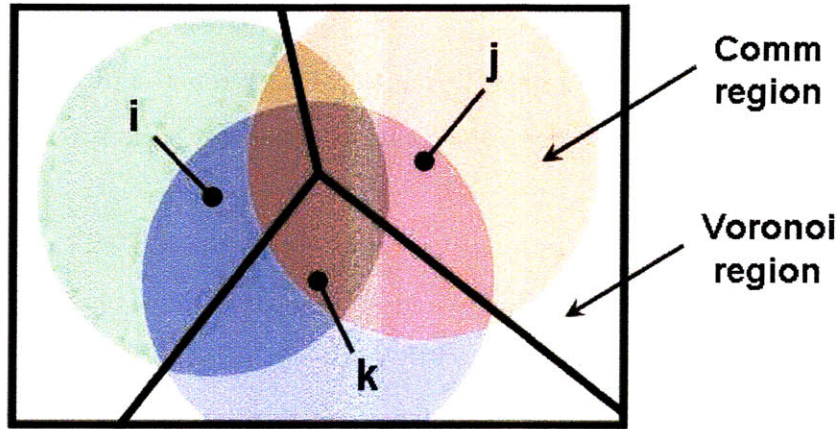


Figure 4-1: A simple example where robots  $i$  and  $j$  share a Voronoi boundary but cannot communicate their state estimates directly. This problem is easily resolved using a mobile ad-hoc network topology to route information through robot  $k$ .

move to become spatially connected, the lack of shared data will prevent the robots from learning about their new neighbors. Thus, no new communication links will be established. We are motivated by these serious and unavoidable complications to develop an algorithm that ensures state estimates flow throughout a team of robots.

#### 4.1.1 Broadcast Scheme

Consider  $n$  robots moving in a global space<sup>2</sup>,  $\mathcal{P}$ . Each robot,  $i \in \{1, \dots, n\}$ , knows its current state,  $\vec{p}_i(t) \in \mathcal{P}$ , by some means of measurement (e.g. GPS or visual localization). We propose that each robot maintains a list of state estimates,  $[\vec{p}_1(t_{i1}), \dots, \vec{p}_n(t_{in})]$ , where  $t_{ij}$  denotes a time stamp at which robot  $i$ 's estimate of robot  $j$ 's state was valid. We have that  $t_{ij} \leq t$  and  $t_{ii} = t$ . Each robot's state estimate is initialized to infinity to indicate that a valid estimate is lacking, except for its own state which is always current.

We desire to communicate state estimates throughout the robot network. For simplicity, we use Time Division Multiple Access (TDMA)<sup>3</sup> to divide the data stream

<sup>2</sup>Although it is easiest to think of the space being  $\mathbb{R}^2$  or  $\mathbb{R}^3$ , the strategy we describe is equally useful with more detailed state estimates (e.g. velocity, acceleration, joint positions, state machine information, etc.)

<sup>3</sup>In this chapter we primarily discuss implementing the proposed strategy using TDMA; however, many other channel access methods are appropriate (e.g. FDMA or CDMA).

into time slots of equal length,  $l$ . During a time slot, one assigned robot is allowed to broadcast over the shared frequency channel. Other robots broadcast one after the other in a predetermined order. One complete broadcast cycle is referred to as a frame.

To broadcast its own state estimate once per frame, the robot's time slot must be long enough to transmit the estimate and an associated time stamp. Such a time slot is considered to have length of  $l = 1$ . Clearly time slots of unit length are not sufficient to transmit information throughout the network; each robot would only be updated with the state estimate of its neighbors with direct communication on the network. For multi-hop networking, the robots need longer time slots to broadcast the estimates of other robots.

One naive strategy is to assign a time slot length equal to the number of robots,  $l = n$ , so that each robot can broadcast its entire list of state estimates, thus creating a simple flooding scheme. Robots that are adjacent on the network use this information to update their own list, retaining only the most current state estimates. The process is repeated for each time slot, naturally propagating state estimates throughout the network without the need of a complicated routing protocol.

Although simple to implement, this strategy is not scalable for a large number of robots. Consider the rate a system can cycle through all time slots to complete one frame. This frame rate,  $r_f$ , gives insight into how quickly state estimates are being forwarded, and therefore how confident distributed controllers can be in using the estimates. For a network of fixed baud rate,  $r_b$ , the maximum frame rate<sup>4</sup> is given by

$$\max(r_f) = \frac{r_b}{lns} \quad (4.1)$$

where  $s$  is the data size of a state estimate and its associated time stamp. For  $l = n$ , increasing the number of robots in the system will decrease the frame rate *quadratically*. This inherent trade-off provides motivation to reduce the length of the time slot. However, if a robot cannot broadcast *all* state estimates within one time

---

<sup>4</sup>We are ignoring overhead associated with TDMA (e.g. guard periods, checksums, etc.)

slot, which estimates are considered more important to broadcast?

### 4.1.2 Importance Function

Many distributed controllers are dependent on spatial relationships between robots. When selecting which state estimate to broadcast, the selection process should also depend on these same spatial relationships. This makes sense because a robot's state is more likely to be useful to controllers in proximity. However, it cannot be considered useless to controllers that are distant due to the mobile nature of the system. We propose that the importance of robot  $i$  broadcasting robot  $j$ 's state estimate is inversely proportional to the distance between robot states.

Since the robots only have access to the state estimates they receive, the following importance function,  $g : \mathcal{P} \times \mathcal{P} \mapsto (0, \infty]$ , uses a distance estimate to give

$$g_{ij}(t) = d(\vec{p}_i(t), \vec{p}_j(t_{ij}))^{-\alpha} \quad (4.2)$$

where  $d : \mathcal{P} \times \mathcal{P} \mapsto [0, \infty)$  is a distance function and  $\alpha \in (0, \infty)$  is a free parameter, both of which are selected for the given distributed controller. For example, a Voronoi coverage controller dependent on linear spatial separation may use a Euclidean distance function with  $\alpha = 1$ . This same distance function is appropriate for a sensor-based controller dependent on light intensity, although  $\alpha = 2$  may be used since light intensity decays quadratically with distance from the source. Conversely, the distance function does not need to be Euclidean or even of continuous topology, such as for truss climbing robots with a finite configuration space. In any case, a robot should consider its own state estimate to be the most important to broadcast. This is reflected in the model since  $g_{ii}$  is infinite for any valid  $d(\cdot, \cdot)$  and  $\alpha$ .

## 4.2 Location-Based Algorithm for Broadcasts

We use the importance function in Equation (4.2) to develop a deterministic algorithm. For a given time slot, this algorithm selects which state estimates a robot

will broadcast. We first describe a probabilistic approach to help formulate the final algorithm.

### 4.2.1 Probabilistic Approach

Consider a robot that needs to select  $l$  state estimates to broadcast during its time slot. We provided motivation in Section 4.1.2 that some selections are more important than others. However, the robot should *not* systematically select the state estimates associated with the highest importance; doing so can prevent estimates from fully dispersing throughout the system. Instead, we propose that the probability of robot  $i$  selecting the state estimate of robot  $j$  is

$$P_{\mathcal{M}_i}^j(t) = \frac{g_{ij}(t)}{\sum_{k \in \mathcal{M}_i} g_{ik}(t)}, \quad j \in \mathcal{M}_i \quad (4.3)$$

where  $\mathcal{M}_i$  is the set of robot indices associated with selectable estimates.

Prior to the first selection for a given time slot,  $\mathcal{M}_i$  is the set of all robot indices. From the full set the robot always selects its own state since it has infinite importance. The robot then removes its index from  $\mathcal{M}_i$  to prevent wasting bandwidth. Since Equation (4.3) is a valid probability mass function, the robot can simply choose the next state estimate at random from the corresponding probability distribution, then remove the corresponding index from  $\mathcal{M}_i$ . This means estimates of closer robots are more likely to be chosen than ones that are farther away. By repeating this process, the entire time slot of length  $l$  can be filled in a straightforward, probabilistic manner.

### 4.2.2 Deterministically Selecting Estimates

It is not ideal in practice to probabilistically select which state estimates to broadcast. Consecutive selections of a particular robot index can be separated by an undesirably long period of time, especially concerning distant robots. By developing a location-based deterministic algorithm, we can increase the average rate at which all state estimates of a given time stamp will propagate throughout a team. In the deterministic case, propagation time is bounded above by the longest path taken among the

---

**Algorithm 1** Deterministic Method for Selecting State Estimates

---

$n$  is the number of robots in the system and  $l$  is the time slot length.

**Require:** Robot  $i$  knows its state  $\vec{p}_i(t)$  and the state estimate of other robots  $\vec{p}_j(t_{ij})$ .

**Require:** Robot  $i$  knows its running counter  $[c_{i1}, \dots, c_{in}]$ .

```
 $\mathcal{M}_i \leftarrow \{1, \dots, n\}$   
for 1 to  $l$  do  
   $P_{\mathcal{M}_i}^{ij}(t) \leftarrow \frac{g_{ij}(t)}{\sum_{k \in \mathcal{M}_i} g_{ik}(t)}, \forall j \in \mathcal{M}_i$   
   $c_{ij} \leftarrow c_{ij}[1 - P_{\mathcal{M}_i}^{ij}(t)], \forall j \in \mathcal{M}_i$   
   $k \leftarrow \arg \max_{k \in \mathcal{M}_i} (c_{ik})$   
   $\mathcal{M}_i \leftarrow \mathcal{M}_i \setminus \{k\}$   
   $c_{ik} \leftarrow 1$   
end for  
return  $\{1, \dots, n\} \setminus \mathcal{M}_i$ 
```

---

estimates. No such bound exists in the probabilistic case, resulting in a positively skewed distribution of propagation times and a larger mean.

We propose that each robot maintains a list of counters,  $[c_{i1}, \dots, c_{in}]$ , which are initially set to a value of one. Using the probability mass function in Equation (4.3), each counter represents the probability that the corresponding index has *not* been selected. Consider a robot's first selection, which will always be its own index. The probability,  $P_{\mathcal{M}_i}^{ii}(t)$ , of selecting index  $i$  is equal to one, while all other probabilities,  $P_{\mathcal{M}_i}^{ij}(t)$  subject to  $j \neq i$ , are equal to zero. This implies that the counter  $c_{ii}$  is multiplied by  $[1 - P_{\mathcal{M}_i}^{ii}(t)] = 0$ , or a zero probability of not being selected, while all other counters,  $c_{ij}$ , are multiplied  $[1 - P_{\mathcal{M}_i}^{ij}(t)] = 1$ , or a probability of one. By selecting the index with the lowest counter value, we are deterministically guiding our method to behave according to the probability distribution described by Equation (4.3). The selected index (in this case  $i$ ) is removed from the set  $\mathcal{M}_i$ , and its corresponding counter ( $c_{ii}$ ) is reset to a value of one. This process is iteratively applied to completely fill a time slot with  $l$  state estimates, with counters maintaining their values between frames. The complete deterministic strategy is given in Algorithm 1.

**Proposition 1 (Deterministic Strategy Time Complexity)** *The time complexity of Algorithm 1 is*

$$T(n, l) = O(nl) \tag{4.4}$$

where  $n$  is the number of robots and  $l$  is the time slot length.

**Proof 1** Assuming  $l < n$ , calculating (4.3) for all  $j \in \mathcal{M}_i$  takes  $O(n)$  time. Since all other loop operations are less than or equal to  $O(n)$  time, Algorithm 1 is of  $O(nl)$  time.

**Remark 1** The time complexity of Algorithm 1 is bounded above by  $O(n \log(n))$  time. This case is for simple flooding when  $l = n$ , resulting in loop operations of less than or equal to  $O(\log(n))$  time.

## 4.3 Network Simulations

We provide insight into the performance of the location-based algorithm in two ways: we conducted Monte Carlo simulations for 100 stationary robots, and we used the embedded control module to simulate nine robots running a distributed coverage algorithm. We first describe the Monte Carlo simulations used to measure information propagation throughout the robot team. Propagation time is the main performance metric for the algorithm. This metric depends on the length of the time slot, or in other words, the number of state estimates communicated during one robot broadcast. We compare these results to the case when the time slot length equals the number of robots, since allowing robots to broadcast every state estimate is the simplest multi-hop scheme. This scheme is referred to as simple flooding.

### 4.3.1 Monte Carlo Simulations

In a MATLAB environment, we simulated a team of 100 stationary robots arranged in a  $10 \times 10$  square grid. Each robot, initialized knowing only its own state estimate, was able to receive broadcasts from its adjacent neighbors along the vertical and horizontal directions. Each robot ran Algorithm 1 in distributed fashion. Over 1000 Monte Carlo simulations were executed for time slots of varying lengths, with each run having a random order for the time slot assignments. For the  $2 \times 2$ ,  $4 \times 4$ ,  $6 \times 6$ ,

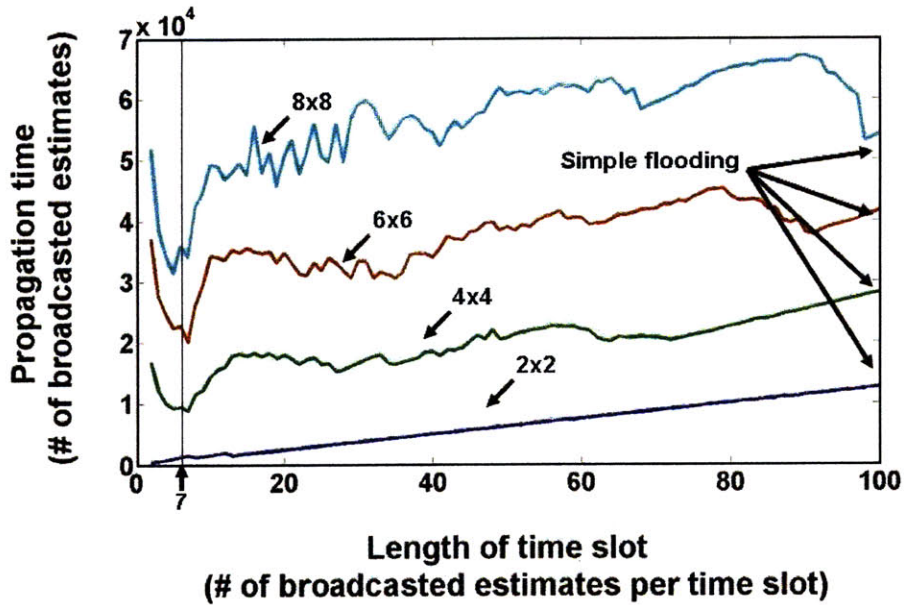


Figure 4-2: This figure shows the average propagation time for the location-based algorithm running on a  $10 \times 10$  stationary robot grid. Averages were taken over 1000 Monte Carlo simulations. For small subgraphs (i.e.  $2 \times 2$ ), update rates of state estimates increased with decreasing time slot lengths. For larger subgraphs, the optimal length was around  $m = 7$ .

and  $8 \times 8$  subgraphs centered on the  $10 \times 10$  graph, we measured the time it took for all subgraph members to exchange state estimates.

Figure 4-2 plots average propagation time for the Monte Carlo simulations. For the smallest subgraph (i.e.  $2 \times 2$ ), state estimates propagated faster with smaller time slot lengths. This relationship makes sense since we are maximizing the frame rate, thus increasing update rates for the local state estimates of highest importance. As the subgraph size increases, very small time slot lengths become less effective at propagating estimates, especially between robots at opposite sides of the subgraph. By using a slightly larger time slot length, a significant improvement in performance over simple flooding is obtained; propagation times for all subgraphs decreased by more than 47% using a time slot length of  $m = 7$ . Analyzing such Monte Carlo plots provides a heuristic technique for selecting an acceptable time slot length for a given control problem.



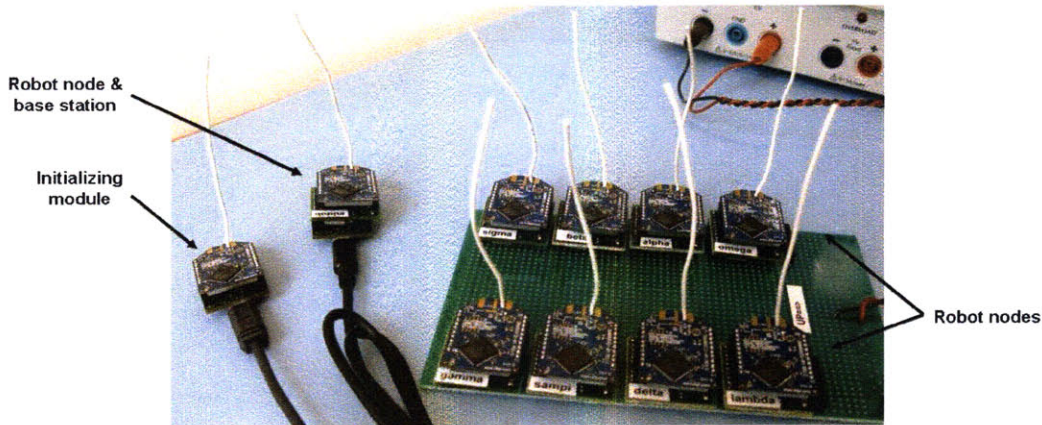


Figure 4-3: A testbed for simulating distributed algorithms using the embedded control module and XBee XSC. Each module locally ran a Voronoi coverage controller and wirelessly communicated state information to other modules in the virtual robot team. A communication range was implemented to add complexity to the wireless network.

### 4.3.2 Voronoi Simulations on the Embedded Control Module

We then tested the algorithm in a simulated robot scenario using the embedded control modules as shown in Figure 4-3. We implemented a Voronoi coverage controller [16] on nine modules, each using a 900 MHz XBee XSC module to wirelessly broadcast state estimates during its assigned time slot. Each control module simulated the dynamics of a flying robot, creating a virtual distributed robot team. In addition, a communication range was implemented such that packets from “out-of-range” robots were automatically dropped. We investigate the performance of the location-based algorithm in a simple scenario where nine virtual robots were tasked to cover a square area. For this scenario the optimal configuration is for the robots to be arranged in a  $3 \times 3$  square grid, thus minimizing the total cost of the Voronoi cost function.

For the location-based algorithm, a time slot length of  $l = 3$  was selected using the Monte Carlo technique previously discussed. We also selected the Euclidean distance function with  $\alpha = 1$  given that the Voronoi coverage controller is linearly dependent on such distance. Each state estimate for the virtual flying robot is constructed of six 32-bit integers (robot identification, time stamp, latitude, longitude, altitude,



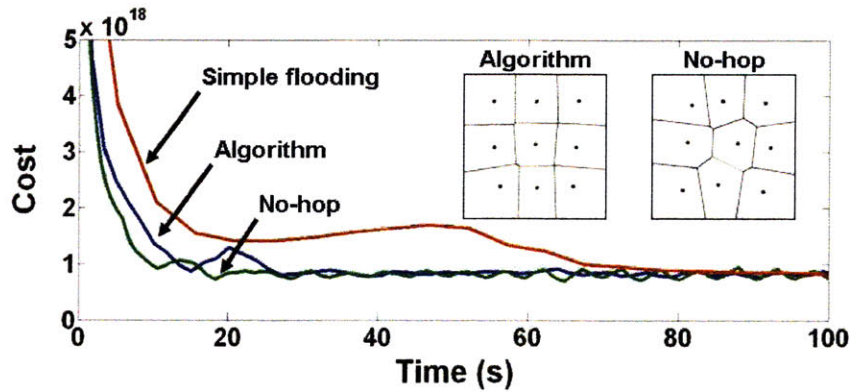


Figure 4-4: Coverage costs are shown for a nine robot system simulated on the embedded control module running a Voronoi coverage controller. The system has a frame rate of 1.7 Hz when using a no-hop scheme ( $l = 1$ ). The system initially performs well, but its inability to multi-hop state estimates resulted in a suboptimal final configuration that does not minimize the Voronoi cost function. A simple flooding scheme ( $l = 9$ ) improved steady state performance, however, the slow frame rate of 0.2 Hz caused the system to initially oscillate in a high cost configuration. The location-based algorithm with a time slot of length  $l = 3$  performed the best overall by combining fast update rates with multi-hop capabilities. The final Voronoi configurations for the algorithm and no-hop simulations are also shown.

and yaw), resulting in a data size,  $s$ , of 192 bits. Given that the wireless hardware could reliably operate at 3000 baud, the resulting frame rate was about 0.6 Hz. For comparison, the simple flooding ( $l = 9$ ) and no-hop ( $l = 1$ ) schemes ran at about 0.2 Hz and 1.7 Hz, respectively. Figure 4-4 shows the resulting coverage cost profiles from these simulations. The location-based algorithm had better initial performance than the simple flooding scheme and better steady state performance than the no-hop scheme. The final Voronoi configurations for the algorithm and no-hop simulations are also shown.



# Chapter 5

## Optimal Coverage with Downward Facing Cameras

As discussed in Chapter 1, there are many applications for a team of collaborating quad-rotor robots. In this chapter<sup>1</sup>, we introduce a novel algorithm that optimally covers a bounded environment using a team of quad-rotor flying robots with downward facing cameras.

We are motivated by a basic information content principle for digital cameras: minimal information per pixel. Using this metric allows for the incorporation of physical, geometric, and optical parameters to give a cost function that represents how well multiple cameras cover a given environment. For our application, we simplify the coverage problem in that the cameras are mounted to the robots in a downward facing orientation. The simplified cost function leads to a gradient-based distributed control algorithm that positions the robots into an optimal configuration.

### 5.1 Optimal Camera Placement

We desire to cover a bounded environment,  $Q \subset \mathbb{R}^2$ , with the fields of view from a number of cameras. We assume  $Q$  is planar, without topography, to avoid the complications of changing elevation or occlusions. Let  $\vec{p}_i^c \in \mathcal{P}_c$  represent the state of

---

<sup>1</sup>The majority of this chapter was published in [52]

a camera, where the state-space,  $\mathcal{P}_c$ , will be characterized later in Section 5.1.1. Each camera is mounted to a robot moving in the global configuration space,  $\mathcal{P}$ . We want to control  $n$  robots in distributed fashion such that their camera placement minimizes the aggregate information per camera pixel over the environment,

$$\min_{(\vec{p}_1^c, \dots, \vec{p}_n^c) \in \mathcal{P}_c^n} \int_Q \frac{\text{info}}{\text{pixel}} d\vec{q}. \quad (5.1)$$

This metric makes sense because the pixel is the fundamental information capturing unit of the camera. Consider the patch of image that is exposed to a given pixel. The information in that patch is reduced by the camera to a low-dimensional representation (i.e. mean color and brightness over the patch). Therefore, the less information content the image patch contains, the less information will be lost in its low-dimensional representation by the pixel. Furthermore, we want to minimize the accumulated information loss due to pixelation over the whole environment  $Q$ , hence the integral.

### 5.1.1 Single Camera

Information per pixel can be represented as the product of two functions,  $f : \mathcal{P}_c \times Q \mapsto (0, \infty]$ , which gives the area in the environment seen by one pixel, and  $\Phi : Q \mapsto (0, \infty)$  which gives the information per area in the environment. The form of  $f(\vec{p}_i^c, \vec{q})$  will be derived from the optics of the camera and geometry of the environment. As shown in Figure 5-1, the function  $\Phi(\vec{q})$  is a positive weighting of importance over  $Q$  and should be specified beforehand (it can also be learned from sensor data, as in [53]). For instance, if all points in the environment are equally important,  $\Phi(\vec{q})$  should be constant over  $Q$ . If some known area in  $Q$  requires more resolution, the value of  $\Phi(\vec{q})$  should be larger in that area. This gives the cost function

$$\min_{\vec{p}_i^c \in \mathcal{P}_c} \int_Q f(\vec{p}_i^c, \vec{q}) \Phi(\vec{q}) d\vec{q}, \quad (5.2)$$

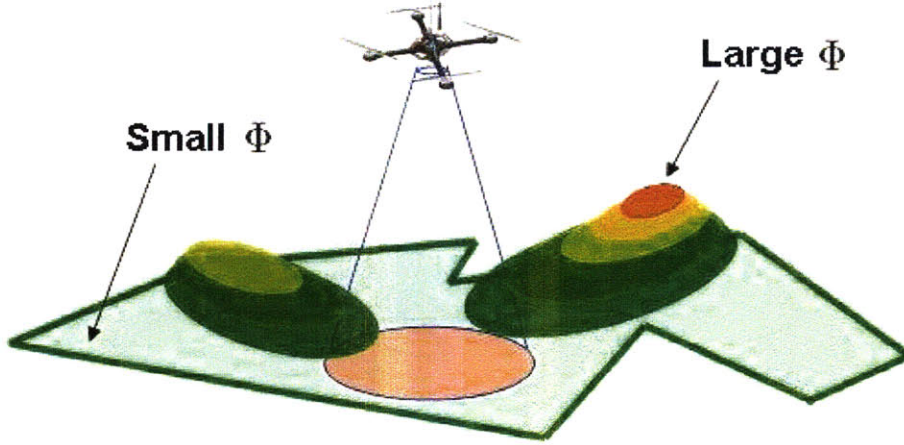


Figure 5-1: An example importance profile is shown for this environment. An larger  $\Phi$  implies that a particular area is of more importance. This figure also illustrates how changes in surface elevations can be represented even though the environment is in  $\mathbb{R}^2$ .

which is of a general form common in the locational optimization and optimal sensor deployment literature [12, 18].

The state of a camera,  $\vec{p}_i^c$ , consists of all parameters associated with the camera that effect the area per pixel function,  $f(\vec{p}_i^c, \vec{q})$ . We consider the special case in which the camera is mounted to the robot in a downward facing orientation. The resulting relationship between the camera and robot state is  $\vec{p}_i^c = [x_i, y_i, z_i]^T$ , giving a camera state space  $\mathcal{P}^c = \mathbb{R}^3$ . Here,  $z_i$  is the height of the camera above the environment,  $Q$ , and  $\vec{v}_i = [x_i, y_i]^T$  is the center point of the field of view,  $\mathcal{B}_i$ . For now, we define a circular field of view to be the intersection of the cone whose vertex is the focal point of the camera lens with the subspace that contains the environment, as shown in Figure 5-2. We have

$$\mathcal{B}_i = \left\{ \vec{q} : \frac{\|\vec{q} - \vec{v}_i\|}{z_i} \leq \tan(\beta^i) \right\} \quad (5.3)$$

where  $\beta^i$  is the half-angle of view of the camera.

To find the area per pixel function,  $f(\vec{p}_i^c, \vec{q})$ , consider the geometry in Figure 5-2. Let  $b$  be the focal length of the lens and  $b^2/(b - z_i)^2$  be the area magnification factor

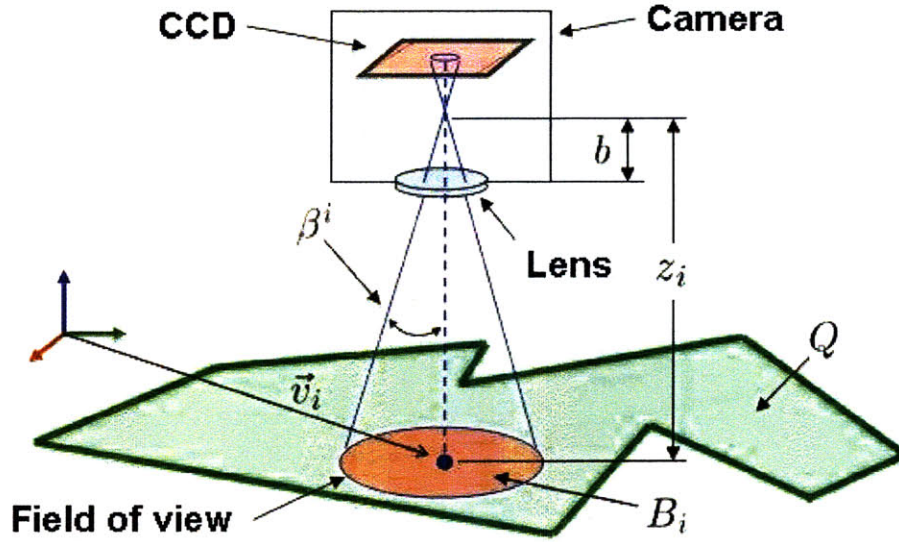


Figure 5-2: This figure shows optical and geometric parameters used in the coverage algorithm. For now, we are assuming a circular field of view.

as defined from classical optics [29]. Inside  $\mathcal{B}_i$ , the area/pixel is equal to the inverse of the area magnification factor times the area of one pixel. Define  $a$  to be the area of one pixel divided by the square of the focal length of the lens. We have,

$$f(\vec{p}_i^c, \vec{q}) = \begin{cases} a(b - z_i)^2 & \text{for } \vec{q} \in \mathcal{B}_i \\ \infty & \text{otherwise,} \end{cases} \quad (5.4)$$

Outside of the field of view, there are no pixels, therefore the area per pixel is infinite (we will avoid dealing with infinite quantities in the multi-camera case). The optimal solution in this simple scenario is for  $\vec{p}_i^c$  to be such that the field of view is the smallest ball that contains  $Q$ . However, with multiple cameras, the problem becomes more challenging.

### 5.1.2 Multiple Cameras

Consider a point  $\vec{q}$  that appears in the image of  $n$  different cameras, such as in Figure 5-3. The number of pixels per area at that point is the sum of the pixels per area for each camera. If we assume all cameras are identical and thus use the same function



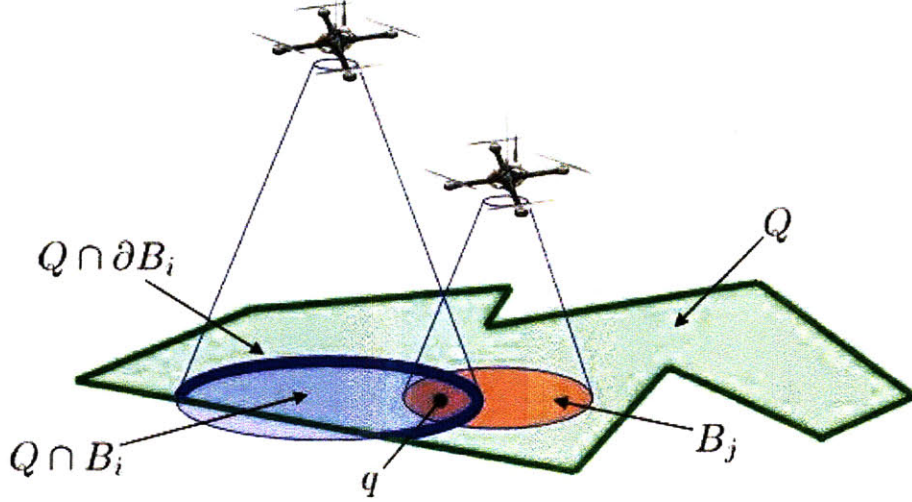


Figure 5-3: An environment can have overlapping fields of view from two or more cameras. The relevant quantities involved in characterizing this scenario are defined.

$f(\vec{p}_i^c, \vec{q})$ , the area per pixel at point  $\vec{q}$  is given by the *inverse* of the sum of the *inverse* of the area per pixel for each camera, or

$$\frac{\text{area}}{\text{pixel}} = \left( \sum_{i=1}^n f(\vec{p}_i^c, \vec{q})^{-1} \right)^{-1}, \quad (5.5)$$

where  $\vec{p}_i^c$  is again the state of the  $i$ th camera. We emphasize that it is the *pixels per area* that sum because of the multiple cameras, not the *area per pixel*. In the overlap region, multiple pixels are observing the same area, resulting in the inverse of the sum of inverses. Incidentally, this is similar to sensor fusion when one combines the variances of multiple noisy measurements.

Finally, we introduce a prior area per pixel,  $w \in (0, \infty)$ . The interpretation of the prior is that there is some pre-existing photograph of the environment (e.g. an initial reconnaissance photograph) from which we can get a base-line area per pixel measurement. This pre-existing information can be arbitrarily vague such that  $w$  can be arbitrarily large, but this prior must exist. The prior also has the benefit of making the cost function finite for all robot positions, being combined with the

camera sensors to get

$$\frac{\text{area}}{\text{pixel}} = \left( \sum_{i=1}^n f(\vec{p}_i^c, \vec{q})^{-1} + w^{-1} \right)^{-1}, \quad (5.6)$$

Let  $\mathcal{N}_{\vec{q}} = \{i : \vec{q} \in \mathcal{B}_i\}$  be the set of indices of cameras for which  $f(\vec{p}_i^c, \vec{q})$  is bounded.

We can now write the area per pixel function as

$$h_{\mathcal{N}_{\vec{q}}}(\vec{p}_1^c, \dots, \vec{p}_n^c, \vec{q}) = \left( \sum_{i \in \mathcal{N}_{\vec{q}}} f(\vec{p}_i^c, \vec{q})^{-1} + w^{-1} \right)^{-1}. \quad (5.7)$$

to give the cost function

$$\mathcal{H}(\vec{p}_1^c, \dots, \vec{p}_n^c) = \int_Q h_{\mathcal{N}_{\vec{q}}}(\vec{p}_1^c, \dots, \vec{p}_n^c, \vec{q}) \Phi(\vec{q}) d\vec{q}. \quad (5.8)$$

which is valid for any area per pixel function  $f(\vec{p}_i^c, \vec{q})$ , and for any camera state space  $\mathcal{P}^c$ .

## 5.2 Distributed Control

We introduce the multi-camera optimization problem

$$\min_{(\vec{p}_1^c, \dots, \vec{p}_n^c) \in \mathcal{P}^n} \mathcal{H}. \quad (5.9)$$

to minimize the total cost of the robot system. By taking the gradient of the cost function  $\mathcal{H}(\vec{p}_1^c, \dots, \vec{p}_n^c)$  with respect to a robot's position  $\vec{p}_i^c$ , we have

$$\frac{\partial \mathcal{H}}{\partial \vec{v}_i} = \int_{Q \cap \partial \mathcal{B}_i} (h_{\mathcal{N}_{\vec{q}}} - h_{\mathcal{N}_{\vec{q}} \setminus \{i\}}) \frac{(\vec{q} - \vec{v}_i)}{\|\vec{q} - \vec{v}_i\|} \Phi(\vec{q}) d\vec{q}, \quad (5.10)$$

and

$$\frac{\partial \mathcal{H}}{\partial z_i} = \int_{Q \cap \partial \mathcal{B}_i} (h_{\mathcal{N}_{\vec{q}}} - h_{\mathcal{N}_{\vec{q}} \setminus \{i\}}) \Phi(\vec{q}) \tan(\beta^i) d\vec{q} - \int_{Q \cap \mathcal{B}_i} \frac{2h_{\mathcal{N}_{\vec{q}}}^2}{a(b - z_i)^3} \Phi(\vec{q}) d\vec{q}. \quad (5.11)$$



where  $\mathcal{N}_{\bar{q}} \setminus \{i\}$  is the set of all indices in  $\mathcal{N}_{\bar{q}}$  except for  $i$ . Refer to [52] for proof and remarks.

### 5.2.1 Control Law

In addition to knowledge about the environment, a robot calculating the gradient of its cost function needs state information about itself and other robots whose field of view overlaps with its own. This suggests a minimal network graph in which all robots  $i$  are connected to all other robots  $j \in \mathcal{N}_i$ , where  $\mathcal{N}_i = \{j \mid Q \cap \mathcal{B}_i \cap \mathcal{B}_j \neq \emptyset, i \neq j\}$ . One interpretation would be to assume that robots with overlapping views of view can communicate in a peer-to-peer fashion. For now we assume this topology, but as we discussed in Chapter 4 this assumption cannot always be realized in practice.

We propose a controller that moves a robot in the opposite direction of its locally calculated gradient component. For all robots, we have

$$\vec{u}_i^c = -k \frac{\partial \mathcal{H}}{\partial \vec{p}_i^c} \quad (5.12)$$

where  $\vec{u}_i^c$  is the control input for robot  $i$  and  $k \in (0, \infty)$  is a control gain. Assuming integrator dynamics for the robots,

$$\dot{\vec{p}}_i^c = \vec{u}_i^c, \quad (5.13)$$

we can prove the convergence of this controller to locally minimize the aggregate information per area. Refer to [52] for proof and remarks.

### 5.2.2 Rectangular Field of View

Until now we have assumed a circular field of view, which eliminates a rotational degree of freedom. Actual cameras have a rectangular CCD array, and therefore a rectangular field of view. Thus, we need to revisit the optimization problem (5.9) and recalculate for a rectangular field of view and a robot with a rotational degree of freedom.

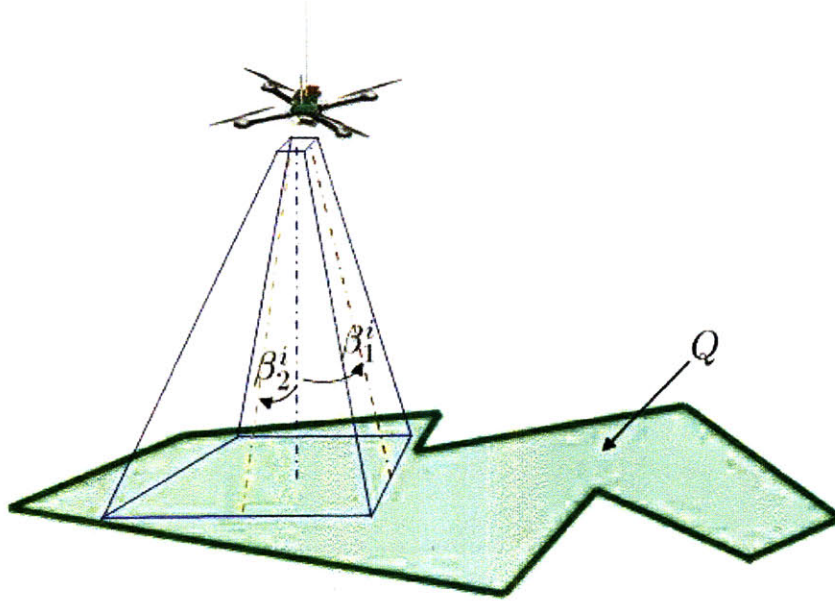


Figure 5-4: The geometry of a camera with a rectangular field of view is shown in this figure.

Let the state space of  $\vec{p}_i^c = [\vec{v}_i^T \quad z_i \quad \psi_i]^T$  be  $\mathcal{P}_c = \mathbb{R}^3 \times \mathfrak{S}$ , where  $\psi_i$  is the yaw angle. Define a rotation matrix

$$\mathbf{R}_c(\psi_i) = \begin{bmatrix} \cos(\psi_i) & \sin(\psi_i) \\ -\sin(\psi_i) & \cos(\psi_i) \end{bmatrix} \quad (5.14)$$

where  $\mathbf{R}_c(\psi_i)\vec{q}$  rotates a vector  $\vec{q}$  expressed in the global coordinate frame, to a coordinate frame aligned with the axes of the rectangular field of view. As is true for all rotation matrices,  $\mathbf{R}_c(\psi_i)$  is orthogonal, meaning  $\mathbf{R}_c(\psi_i)^T = \mathbf{R}_c(\psi_i)^{-1}$ . Using this matrix, define the field of view of robot  $i$  to be

$$\mathcal{B}_i = \left\{ \vec{q} : |\mathbf{R}_c(\psi_i)(\vec{q} - \vec{v}_i)| \leq z_i \tan(\vec{\beta}^i) \right\} \quad (5.15)$$

where  $\vec{\beta}^i = [\beta_1^i, \beta_2^i]^T$  is a vector of the camera's two half-view angles associated with two perpendicular edges of the rectangle, as shown in Figure 5-4. Here the  $\leq$  symbol applies element-wise such that all elements in the vector must satisfy the inequality.

We have to break up the boundary of the rectangle into each of its four edges. Let  $l_k$  be the  $k$ th edge, and define four outward-facing normal vectors  $\vec{n}_k$ , one associated with each edge, where  $\vec{n}_1 = [1, 0]^T$ ,  $\vec{n}_2 = [0, 1]^T$ ,  $\vec{n}_3 = [-1, 0]^T$ , and  $\vec{n}_4 = [0, -1]^T$ .

With a rectangular field of view, the gradient of the cost function,  $\mathcal{H}(\vec{p}_1^c, \dots, \vec{p}_n^c)$ , becomes

$$\frac{\partial \mathcal{H}}{\partial \vec{v}_i} = \sum_{k=1}^4 \int_{Q \cap l_k} (h_{\mathcal{N}_{\vec{q}}} - h_{\mathcal{N}_{\vec{q}} \setminus \{i\}}) \mathbf{R}_c(\psi_i)^T \vec{n}_k \Phi(\vec{q}) d\vec{q} \quad (5.16)$$

$$\frac{\partial \mathcal{H}}{\partial z_i} = \sum_{k=1}^4 \int_{Q \cap l_k} (h_{\mathcal{N}_{\vec{q}}} - h_{\mathcal{N}_{\vec{q}} \setminus \{i\}}) \tan(\theta)^T \vec{n}_k \Phi(\vec{q}) d\vec{q} - \int_{Q \cap \mathcal{B}_i} \frac{2h_{\mathcal{N}_{\vec{q}}}^2}{a(b-z_i)^3} \Phi(\vec{q}) d\vec{q} \quad (5.17)$$

$$\frac{\partial \mathcal{H}}{\partial \psi_i} = \sum_{k=1}^4 \int_{Q \cap l_k} (h_{\mathcal{N}_{\vec{q}}} - h_{\mathcal{N}_{\vec{q}} \setminus \{i\}}) \cdot (\vec{q} - \vec{v}_i)^T \mathbf{R}_c(\psi_i + \pi/2)^T \vec{n}_k \Phi(\vec{q}) d\vec{q} \quad (5.18)$$

Refer to [52] for proof and remarks.

### 5.2.3 Discrete Controllers

To be implemented on the embedded control module, the integrals in the controller must be computed using a discretized approximation. Let  $\widehat{Q}$  be a discretized set of  $m$  points uniformly spaced  $\Delta_{\widehat{Q}}$  units apart on a square grid. Given  $\vec{p}_i^c$ , let  $\widehat{\partial \mathcal{B}_i}$  be a discretized set of  $m$  points uniformly spaced  $\Delta_{\widehat{\partial \mathcal{B}_i}}$  units apart on  $\partial \mathcal{B}_i$ . We select  $\widehat{Q}$  and  $\widehat{\partial \mathcal{B}_i}$  to best approximate  $Q$  and  $\partial \mathcal{B}_i$ , respectively. Using these sets, we can formulate discrete controllers that approximate Equation (5.12). Algorithms for circular and rectangular fields of view are given in Algorithms 2 and 3, respectively.

**Proposition 2 (Controller Time Complexity - Circular Fields of View)** *The time complexity of Algorithm 2 is*

$$T(n, m) = O(nm) \quad (5.19)$$

---

**Algorithm 2** Discretized Controller for Circular Fields of View

---

$k_v$  and  $k_z$  are lateral and vertical controller gains, respectively

**Require:** Robot  $i$  knows its state  $\vec{p}_i^c$

**Require:** Robot  $i$  knows the environment  $Q$

**Require:** Robot  $i$  knows the information per area function  $\Phi(\vec{q})$ .

**Require:** Robot  $i$  can communicate with all robots  $j \in \mathcal{N}_i$ .

**loop**

Update  $\vec{p}_j^c, \forall j \in \mathcal{N}_i$

$$\frac{\partial \widehat{\mathcal{H}}}{\partial \vec{v}_i} \leftarrow \sum_{q \in Q \cap \widehat{\partial \mathcal{B}}_i} (h_{\mathcal{N}_{\vec{q}}} - h_{\mathcal{N}_{\vec{q}} \setminus \{i\}}) \frac{(\vec{q} - \vec{v}_i)}{\|\vec{q} - \vec{v}_i\|} \Phi(\vec{q}) \Delta_{\widehat{\partial \mathcal{B}}_i}$$

$$\frac{\partial \widehat{\mathcal{H}}}{\partial z_i} \leftarrow \sum_{q \in Q \cap \widehat{\partial \mathcal{B}}_i} (h_{\mathcal{N}_{\vec{q}}} - h_{\mathcal{N}_{\vec{q}} \setminus \{i\}}) \Phi(\vec{q}) \tan(\theta) \Delta_{\widehat{\partial \mathcal{B}}_i} - \sum_{q \in \widehat{Q} \cap \mathcal{B}_i} \frac{2h_{\mathcal{N}_{\vec{q}}}^2}{a(b-z_i)^3} \Phi(\vec{q}) \Delta_{\widehat{Q}}^2$$

$$\vec{v}_i(t + \Delta t) \leftarrow \vec{v}_i(t) - k_v \frac{\partial \widehat{\mathcal{H}}}{\partial \vec{v}_i}$$

$$z_i(t + \Delta t) \leftarrow z_i(t) - k_z \frac{\partial \widehat{\mathcal{H}}}{\partial z_i}$$

**end loop**

---

---

**Algorithm 3** Discretized Controller for Rectangular Fields of View

---

$k_v, k_z,$  and  $k_\psi$  are lateral, vertical, and rotational controller gains, respectively

**Require:** Robot  $i$  knows its state  $\vec{p}_i^c$

**Require:** Robot  $i$  knows the environment  $Q$

**Require:** Robot  $i$  knows the information per area function  $\Phi(\vec{q})$ .

**Require:** Robot  $i$  can communicate with all robots  $j \in \mathcal{N}_i$ .

**loop**

Update  $\vec{p}_j^c, \forall j \in \mathcal{N}_i$

$$\frac{\partial \widehat{\mathcal{H}}}{\partial \vec{v}_i} \leftarrow \sum_{k=1}^4 \sum_{q \in Q \cap \widehat{l}_k} (h_{\mathcal{N}_{\vec{q}}} - h_{\mathcal{N}_{\vec{q}} \setminus \{i\}}) \mathbf{R}_c(\psi_i)^T \vec{n}_k \Phi(\vec{q}) \Delta_{\widehat{\partial \mathcal{B}}_i},$$

$$\frac{\partial \widehat{\mathcal{H}}}{\partial z_i} \leftarrow \sum_{k=1}^4 \sum_{q \in Q \cap \widehat{l}_k} (h_{\mathcal{N}_{\vec{q}}} - h_{\mathcal{N}_{\vec{q}} \setminus \{i\}}) \tan(\theta)^T \vec{n}_k \Phi(\vec{q}) \Delta_{\widehat{\partial \mathcal{B}}_i} - \sum_{q \in \widehat{Q} \cap \mathcal{B}_i} \frac{2h_{\mathcal{N}_{\vec{q}}}^2}{a(b-z_i)^3} \Phi(\vec{q}) \Delta_{\widehat{Q}}^2$$

$$\frac{\partial \widehat{\mathcal{H}}}{\partial \psi_i} \leftarrow \sum_{k=1}^4 \sum_{q \in Q \cap \widehat{l}_k} (h_{\mathcal{N}_{\vec{q}}} - h_{\mathcal{N}_{\vec{q}} \setminus \{i\}}) \cdot (\vec{q} - \vec{v}_i)^T \mathbf{R}_c(\psi_i + \pi/2)^T \vec{n}_k \Phi(\vec{q}) \Delta_{\widehat{\partial \mathcal{B}}_i}$$

$$\vec{v}_i(t + \Delta t) \leftarrow \vec{v}_i(t) - k_v \frac{\partial \widehat{\mathcal{H}}}{\partial \vec{v}_i}$$

$$z_i(t + \Delta t) \leftarrow z_i(t) - k_z \frac{\partial \widehat{\mathcal{H}}}{\partial z_i}$$

$$\psi_i(t + \Delta t) \leftarrow \psi_i(t) - k_\psi \frac{\partial \widehat{\mathcal{H}}}{\partial \psi_i}$$

**end loop**

---

where  $n$  is the number of robots and  $m$  is the number of elements in  $\widehat{Q}$  and  $\widehat{\partial\mathcal{B}_i}$ .

**Proof 2** Discretizing (5.10) and (5.11), we have

$$\sum_{j=1}^m \left( O(1) + \sum_{k=1}^n O(1) \right) \quad (5.20)$$

and

$$\sum_{j=1}^m \left( O(1) + \sum_{k=1}^n O(1) + \sum_{k=1}^{n-1} O(1) \right) \quad (5.21)$$

time, respectively. Summing these results gives  $O(nm)$  time. Since all other loop operations are less than  $O(n)$  time, Algorithm 2 is of  $O(nm)$  time.

**Proposition 3 (Controller Time Complexity - Rectangular Fields of View)**

The time complexity of Algorithm 3 is

$$T(n, m) = O(nm) \quad (5.22)$$

where  $n$  is the number of robots and  $m$  is the number of elements in  $\widehat{Q}$  and  $\widehat{\partial\mathcal{B}_i}$ .

**Proof 3** Time complexity for discretizing (5.16) and (5.17) are identical to (5.20) and (5.21), respectively. Discretizing (5.18), we have

$$\sum_{j=1}^m \left( O(1) + \sum_{k=1}^n O(1) \right) \quad (5.23)$$

time. Summing these results gives  $O(nm)$  time. Since all other loop operations are less than  $O(n)$  time, Algorithm 3 is of  $O(nm)$  time.



# Chapter 6

## Robot Experiments

To demonstrate the functionality of our embedded control module for a team of AscTec Hummingbird flying quad-rotor robots, we conducted numerous experiments in both indoor and outdoor environments. We first implemented the optimal coverage algorithm presented in Chapter 5 indoors in distributed fashion. Three quad-rotor robots optimally positioned their downward facing cameras over a fixed environment. The robot team used the 2.4 Ghz Digi XBee-PRO radio modules to acquire position information from the motion capture system. In preparation for the outdoor extension of this algorithm, we then conducted a mobile ad hoc network experiment above the urban MIT campus using the same hardware configuration. State estimates were routed in multi-hop fashion from the robot team to a ground base station using a simple flooding scheme.

The limited communication range we experienced during the network experiment led to the selection of the 900 Mhz Digi XBee-XSC radio modules for outdoor flights. To accommodate the decrease in baud rate when compared to the XBee-PRO, we developed the location-based multi-hop algorithm described in Chapter 4. We then implemented this algorithm on five quad-rotor flying robots and four ground base stations to demonstrate network adaptability for a dynamic robot team. This same algorithm was used to allow state estimates to propagate during the outdoor deployment of our optimal coverage algorithm.

## 6.1 Optimal Coverage of an Indoor Environment

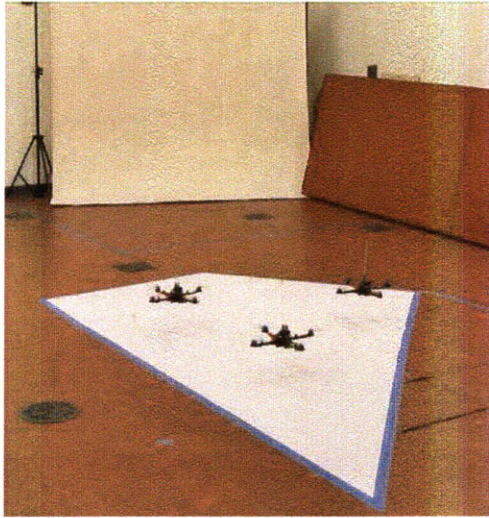
The optimal coverage algorithm for a circular field of view was implemented on our embedded control module, running asynchronously in a fully distributed fashion. The algorithm calculated the waypoints  $\vec{v}_i(t)$  and  $z_i(t)$  from Algorithm 2 at 1 Hz. This time-scale separation between the coverage algorithm and the low level PID controllers was required to approximate the integrator dynamics assumed in Equation (5.13). The camera parameters were set to  $a = 10^{-6}$  and  $b = 10^{-2}$  meters (which are typical for commercially available cameras), the field of view was  $\beta^z = 35^\circ$ , the information per area was a constant  $\phi(q) = 1$ , the prior area per pixel was  $w = 10^{-6}$  meters<sup>2</sup>, and the control gain was  $k = 10^{-5}$ . The environment to be covered was a skewed rectangle, 3.7 meters across at its widest, shown in white in Figure 6.1.

To test the effectiveness of the algorithm and its robustness to robot failures, we conducted experiments as follows: 1) three robots moved to their optimal positions using the algorithm, 2) one robot was manually removed from the environment, and the remaining two were left to reconfigure automatically, 3) a second robot was removed from the environment and the last one was left to reconfigure automatically. Figure 6.1 shows photographs of a typical experiment at the beginning (Figure 6-1(a)), after the first stage (Figure 6-1(b)), after the second stage (Figure 6-1(c)), and after the third stage (Figure 6-1(d)).

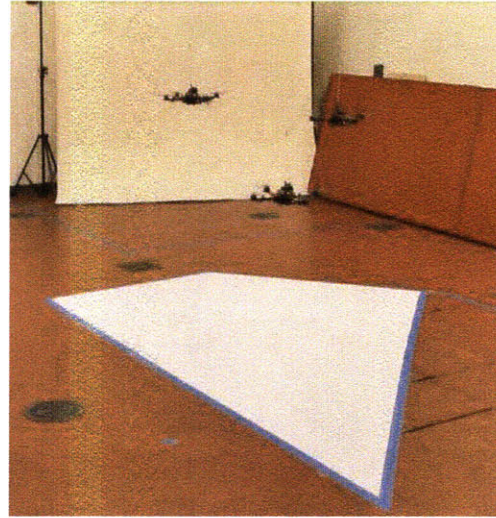
The coverage cost of the robots over the course of the whole experiment, averaged over 19 experiments, is shown in Figure 6-2, where the error bars represent one standard deviation. Notice that when one robot is removed, the cost function momentarily increases, then decrease as the remaining robots find a new optimal configuration. The algorithm proved to be robust to the significant, highly nonlinear unmodeled aerodynamic effects of the robots, and to individual robot failures.

We repeated the above experiment a total of 20 times. Of these 19 were successful, while in one experiment two of the robots collided in mid air. The collision was caused by an unreliable gyroscopic sensor, not by a malfunction of the coverage algorithm. With appropriate control gain values, collisions are avoided by the algorithm's natural

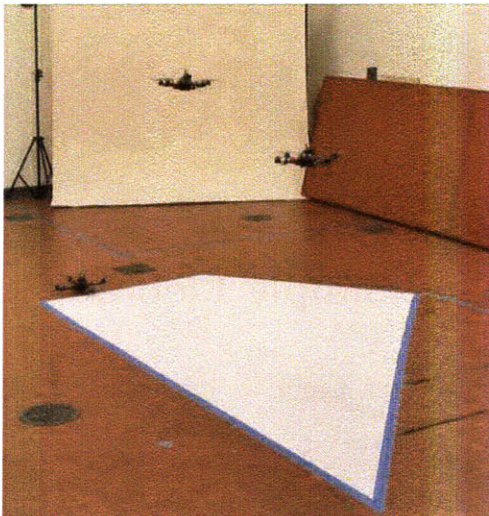




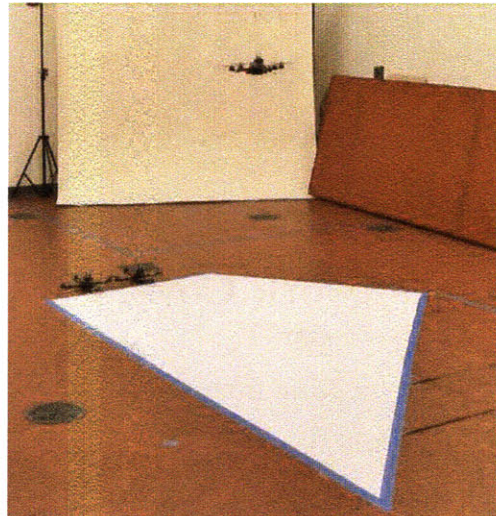
(a) Initial Configuration



(b) Three Robot Configuration



(c) Two Robot Configuration



(d) One Robot Configuration

Figure 6-1: Frame shots from an experiment with three AscTec Hummingbird quadrotor robots are shown. After launching from the ground (Figure 6-1(a)), the three robots stabilize in an optimal configuration (Figure 6-1(b)). Then one robot is manually removed to simulate a failure, and the remaining two move to a new optimal position (Figure 6-1(c)). Finally a second robot is removed and the last one stabilizes at an optimal position (Figure 6-1(d)). The robots move so that their fields of view (which cannot be seen in the snapshots) cover the environment, represented by the white polygon.

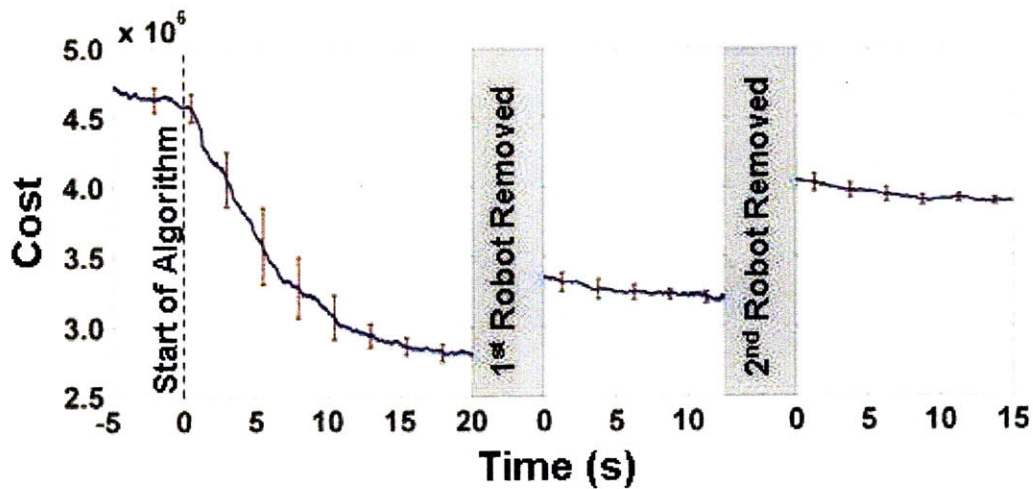


Figure 6-2: The cost function during the three stages of the experiment, averaged over 19 successful experiments, is shown. The error bars denote one standard deviation. The experiments demonstrate the performance of the algorithm, and its ability to adapt to unforeseen robot failures.

tendency for neighbors to repel one another.

## 6.2 Network Coverage Using Simple Flooding

Using the same team from the indoor coverage experiments, we implemented a simple flooding scheme using TDMA in an outdoor urban environment. Our embedded control module acquired GPS coordinates, altitude readings, and temperature measurements from the AutoPilot module at 4 Hz. During a robot's assigned time slot, the 2.4 GHz XBee-PRO module broadcasted the state estimates of all three robots in the system, where states for the other two robots were acquired from previously received broadcasts. Each robot's state is described by an array containing five 32-bit integers (i.e. time stamp, GPS latitude, GPS longitude, altitude reading, and temperature measurement), resulting in a data size of 160 bits. Given that our wireless network could reliably run at a baud rate of 57.6 kbps, the maximum frame rate using Equation 4.1 was about 40 Hz - an order of magnitude faster than what is needed for our coverage algorithm.



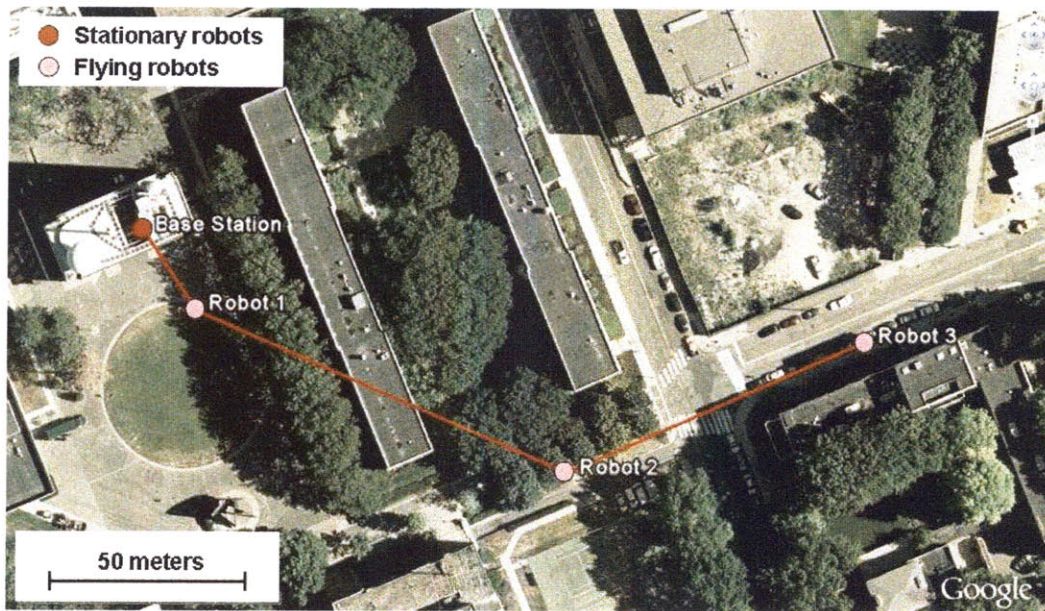


Figure 6-3: An example mobile ad hoc network graph from the quad-rotor flying robot experiment is plotted over an areal image. For a system of three robots, the simple flooding scheme was successful in routing state estimates back to the base station.

A base station monitored network activity as the robots were manually piloted to expand the spatial size of the network. Figure 6-3 shows a plot of the position estimates of the three robots as received by the base station using the full broadcast strategy described. The positions are laid over an aerial photograph of the environment. The robot temperature estimates were also recorded by the base station. Figure 6-4 plots temperature with respect to time for the three robots. In addition, the base station recorded how the estimates were received. If they were received directly from the originating source, the data point is labeled with an  $\times$ . Otherwise, the data traveled a multi-hop route to the base station and is labeled with an  $\circ$ .

### 6.3 Location-Based Algorithm

Although the simple flooding scheme in Section 6.2 was successful in routing state estimates through the robot team, we were disappointed in the communication range

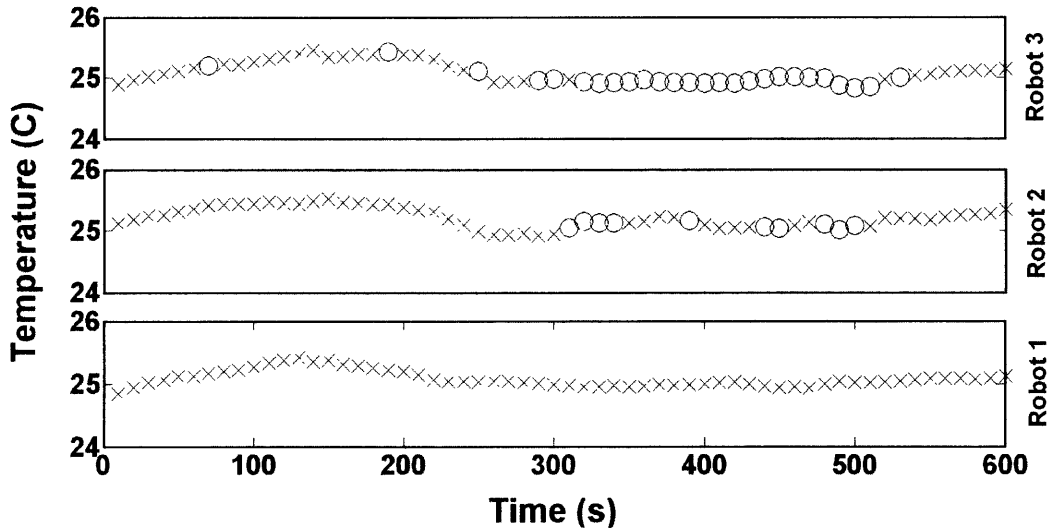


Figure 6-4: This figure shows acquired physical data from the experiment. Robot state estimates, as recorded by the base station, included onboard temperature. Estimates transferred directly to the base station from the originating robot are labeled with  $\times$ 's, while ones needing multiple hops are labeled with  $\circ$ 's.

obtained by the 2.4 Ghz Digi XBee-PRO radio modules. At best peer-to-peer communication links of about 100 m were maintained, although we experienced dropouts between robots located less than 50 meters from each other. To allow for large scale outdoor experiments, we decided to use the longer range 900 Mhz Digi XBee-XSC radio module. However, this hardware change results in the decrease of our wireless baud rate from about 57.6 kbps to 3 kbps. This motivated us to implement the location-based algorithm described in Chapter 4 to effectively multi-hop state estimates throughout the robot team.

This experimental setup was designed to emulate an initial configuration for five flying robots running the downward facing camera coverage controller. In addition, four stationary robots were used to monitor the system as base stations, thus creating a nine robot team. Since the coverage controller has a spatial dependence similar to the Voronoi coverage controller from Section 4.3.2, the same time slot length ( $l = 3$ ), distance function (Euclidean), and  $\alpha (= 1)$  were used. Figure 6-5 shows the network topology of a random deployment configuration prior to starting the coverage controller. Here we limited the communication range to 30 meters; this radio module

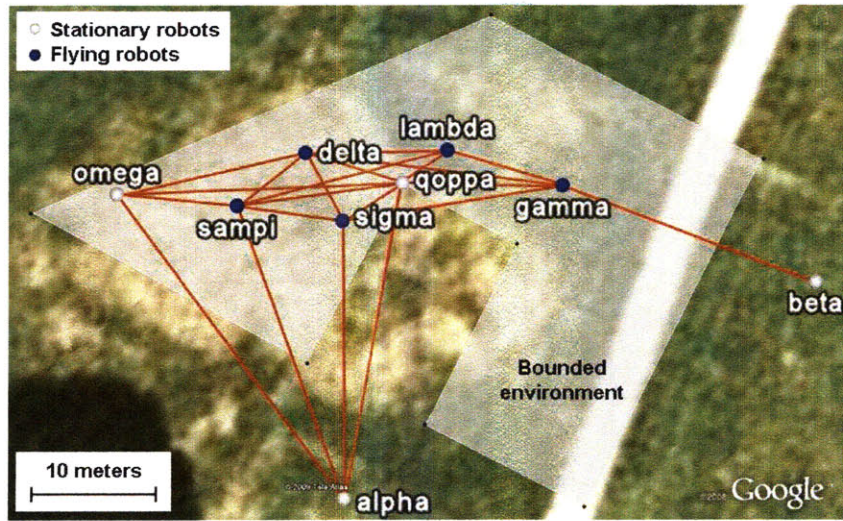


Figure 6-5: An example mobile ad hoc network graph from the quad-rotor flying robot experiment is plotted in Google Earth. For this nine robot system, the location-based algorithm routes state estimates through the entire team. The bounded environment from the downward facing camera coverage problem is also shown.

was able to produce links in excess of 100 meters in separate range tests.

Figure 6-6 plots the time stamp of the most current state estimates as received by the stationary robot beta, which can be considered the “worst case” receiver since it is the most remote robot in the team. As previously discussed, beta’s own state estimate is always considered to be current. Estimates of other robots are updated as they are received by team broadcasts, whether directly from the originating robot or indirectly in a multi-hop fashion. Since closer robots are considered more important in the algorithm formulation, this results in their state estimates being more current with more frequent updates. This characteristic can be seen easily in Figure 6-7, where the state estimates of gamma and lambda are more important to beta than alpha and omega.

## 6.4 Optimal Coverage of an Outdoor Environment

We combined the optimal coverage algorithm in Chapter 5 and the location-based multi-hopping algorithm in Chapter 4 to form a completely autonomous system that



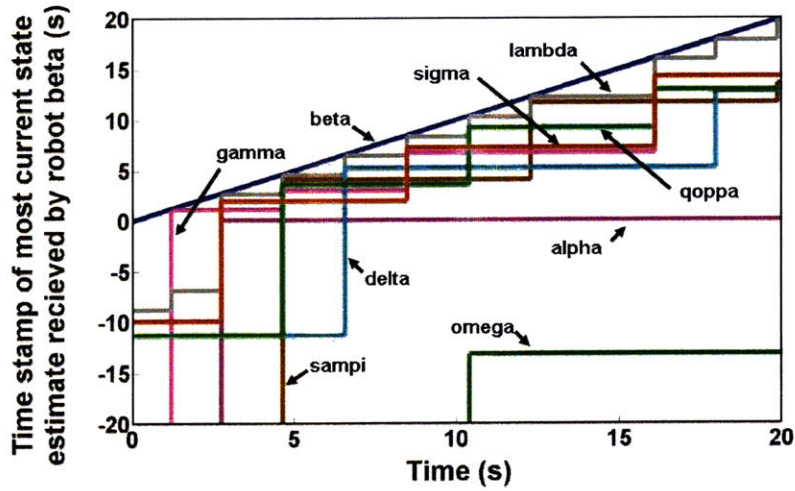


Figure 6-6: This plot shows the time stamp of the most current state estimates received by the stationary robot beta. Estimates of closer, more important robots are updated more frequently and tend to be more current, which validates the location-based algorithm.

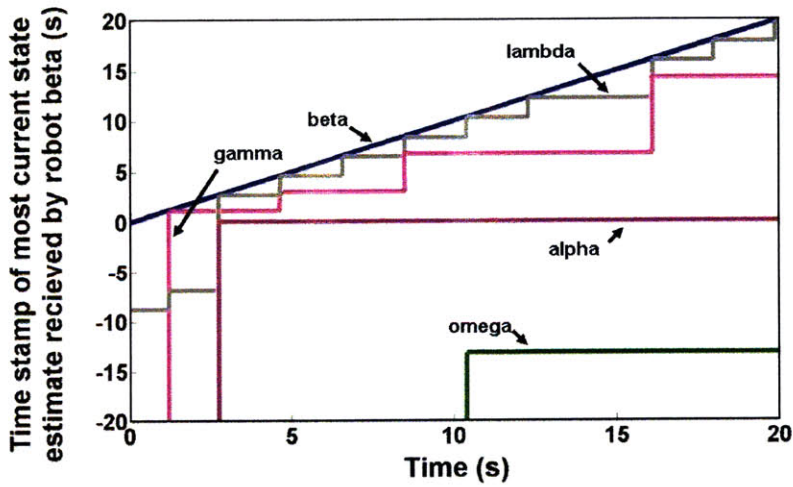


Figure 6-7: A simplified plot of Figure 6-5 is shown. From the frequency of the updates, we can conclude that gamma started the closest to beta but then moved away during the experiment. Once lambda moved into communication range with beta, state estimates of the team began to be routed through lambda.

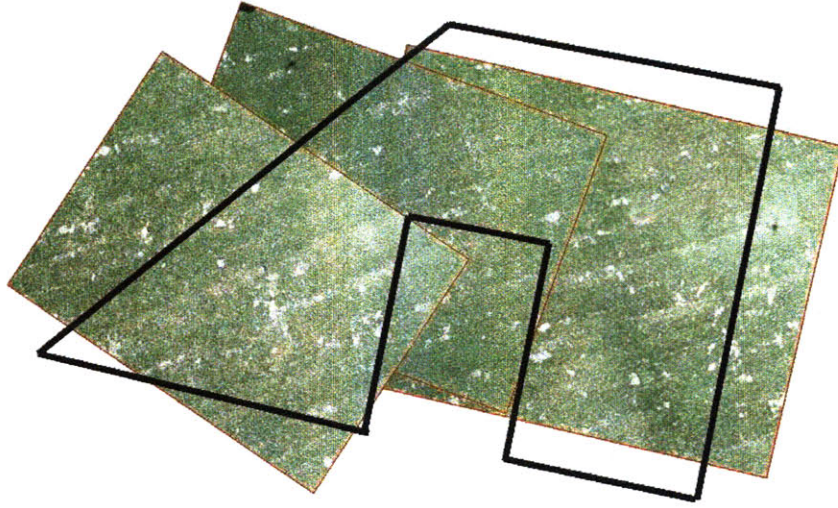


Figure 6-8: The embedded control modules are used to cover an outdoor environment. Off-the-shelf miniature camcorders recorded the field of view of each robot during the experiment.

can be deployed in an outdoor environment. Three quad-rotor flying robots were used to route state information to three stationary robots, forming a six robot team. Compared to the indoor experiment in Section 6.1, the environment was drastically different in two ways: 1) the skewed rectangle, which measured approximately 60 meters at its widest, was much larger; and 2) a square area was removed to create a non-convex environment. However, we were able to use the same camera parameters for the coverage algorithm as before, only this time a rectangular field of view was used with half angles of  $\beta_1^i = 35^\circ$  and  $\beta_2^i = 26.25^\circ$ .

To assist in visualizing the team's coverage, we installed an inexpensive video camera on the bottom of each robot. Figure 6-8 shows an example snapshot of the team's coverage during the experiment. We noted that the mosaic did not fully cover the environment, however we believe this outcome is the result of three factors: 1) the environment area was discretized and may have not represented the perimeter of the skewed rectangle well; 2) there was considerable error in GPS readings, which may have prevent the robots from settling into an optimal configuration; and 3) due to limited amount of battery life, we may have not given the team enough time to fully cover the environment.

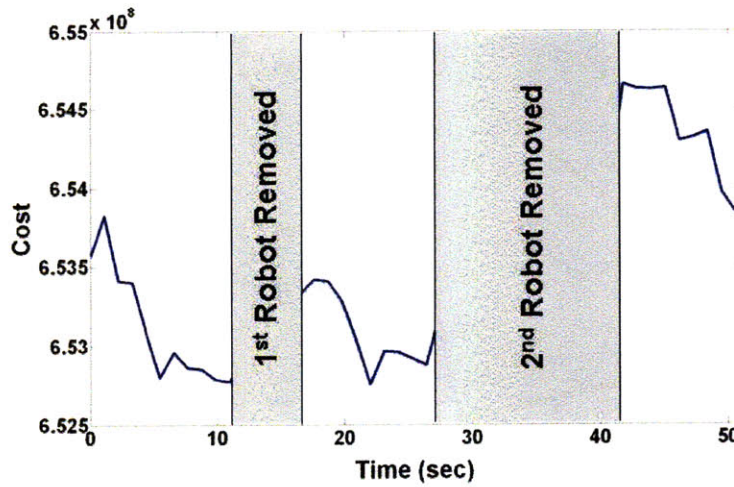


Figure 6-9: The cost function during the three stages of the outdoor experiment. Like the indoor experiment, we demonstrated the performance of the algorithm and its ability to adapt to unforeseen robot failures.

Figure 6-9 shows the cost function during the outdoor experiment. The three flying robots converged to an optimal configuration before one robot was manually removed from the system. This removal process was repeated after the two robot team converged to a new optimal configuration. The ability to adapt to unforeseen robot failures demonstrates the robustness of the algorithm even in outdoor environments. This experiment was repeated once more with similar results.



# Chapter 7

## Conclusions

In this thesis we discussed the distributed control of a high performance quad-rotor flying robot team. This research produced the design for an onboard embedded control module, with which we successfully deployed a distributed robot team in both an indoor and outdoor environment. In both environments we constructed a mobile ad hoc network infrastructure to propagate state estimates among the robots. Our system design was verified in numerous experiments that successfully showed the self-organization of the distributed team.

### 7.1 Lessons Learned

The process of designing, fabricating, programming, and implementing the embedded control module provided an extremely rewarding experience. The following summarizes some lessons learned during our research effort:

1. *Version Control Among All Robots* - Every computer scientist has (or will) learn the importance of version control. This task becomes exponentially more important when considering a distributed system. Careful bookkeeping is needed to insure all robot firmware is current and compatible. We made our biggest mistake when we assumed identical protocol among different firmware versions for the AscTec AutoPilot module. The resulting deployment produced a team

where only half of the robots operated correctly, even though all firmware for the embedded control module was current. This mistake could have been easily avoided (or easily debugged) if we ensured that all hardware in the system was up-to-date<sup>1</sup>.

2. *Scaling Gradually* - Since all hardware and software was designed to scale automatically, we were often tempted to drastically increase the number of robots in the team between deployments. This action was usually less efficient (e.g. time, battery life, etc.) than gradually increasing the scale of the experiment. For example, consider our outdoor coverage experiment in Section 6.4. Even though it doesn't seem like a drastic increase, we wasted a good amount of time by increasing the number of robots from one to three. A two robot experiment would have given insight into deployment strategy for multiple robots. Instead, we had the added complexity of another robot member, which only complicated the situation and extended the setup time before the actual three robot experiment was conducted.
3. *Hardware-In-Loop Simulations* - We relied heavily on MATLAB and C code simulations to initially construct our distributed algorithms. However, we added another simulation step, hardware-in-loop (HIL), to further debug our code prior to the field experiments. By representing the quad-rotor flying robot as a virtual model on the embedded control module, we were able to closely monitor the performance of our algorithms running on the actual hardware. This setup, as shown in Figure 4-3, provided us with an effective tool for software integration and control parameter tuning.

## 7.2 Future Work

The primary motivation for developing the embedded control module was to support the development of a variety of high-dimensional (+3) distributed algorithms. The

---

<sup>1</sup>We would like to thank Daniel Gurdan from Ascending Technologies for helping us identify this incompatibility.

resulting system is advanced in overall functionality yet intuitive and user friendly. Currently we are developing a general user interface to be used by many distributed robot projects. These projects are not necessarily using quad-rotor flying robots - Yun et. al are using our system to propagate state estimates among assembly robots [40]. We hope this effort will form a testbed where a heterogeneous robot team can demonstrate distributed behavior; we are inspired by the work of Jon How et. al and the RAVEN testbed [35].

Concerning our location-based multi-hop algorithm from Chapter 4, we believe the simplicity of this strategy enables the cascading of additional location-based algorithms. The spatial reuse of time slots for robots separated by multiple hops can allow for virtually infinite robot team sizes and spatial coverage. In addition, our algorithm is not limited to TDMA; significant performance gains can be obtained by using different channel access method, such as code division multiple access (CDMA) and frequency division multiple access (FDMA).

Finally, we are actively pursuing more complex experiments for the optimal coverage algorithm with downward facing cameras from Chapter 5. Results for teams of 5+ robots will be published in the near future. We are also exploring the coverage of dynamically changing environments with respect to geometry of the boundary and the shape of the importance function. Our ultimate goal for this algorithm is to seamlessly incorporate robots of different dynamics and cameras of different optical properties.



# Appendix A

## Communication Protocol

The following tables outline the communication protocol used over the wireless network. Sections of the packets relating to byte count and checksum calculations are labeled. For all packets, the checksum section may be “looped;” for example, multiple state estimates may be sent within one state packet. The increase in length is accounted for by both the byte count and the checksum.

Table A.1: The state packet is sent between robots to update their state estimate lists. The first entry in the checksum loop contains the sender’s estimate, which is assumed to be current and is used to synchronize teammates’ clocks.

	<b>Description</b>	<b>Type</b>	<b>Bytes</b>	<b>Defined</b>	
	start string	uint8 [3]	3	">*>"	
	packet indicator	uint8	1	'S'	
	byte count	uint32	4		
begin byte count	robot id	uint32	4		begin checksum
	time stamp	uint32	4	[ $\mu$ s]	
	$x_i$	int32	4	[mm]	
	$y_i$	int32	4	[mm]	
	$z_i$	int32	4	[mm]	
	$\psi_i$	int32	4	[mrad]	end checksum
end byte count	checksum	uint32	4	crc32	

Table A.2: The global packet provides position information for target robots during indoor operation. This information overrides all other methods of position acquisition. This packet has no effect during outdoor operation.

	<b>Description</b>	<b>Type</b>	<b>Bytes</b>	<b>Defined</b>	
	start string	uint8 [3]	3	">*>"	
	packet indicator	uint8	1	'G'	
	byte count	uint32	4		
begin byte count	robot id	uint32	4		begin checksum
	time stamp	uint32	4	[ $\mu$ s]	
	$x_i$	int32	4	[mm]	
	$y_i$	int32	4	[mm]	
	$z_i$	int32	4	[mm]	
	$r_i^x$	int32	4	[mrad]	
	$r_i^y$	int32	4	[mrad]	
	$r_i^z$	int32	4	[mrad]	end checksum
end byte count	checksum	uint32	4	crc32	

Table A.3: The communication packet initializes TDMA communication among the robots in the team. The packet also determines which robots are in the team for high level control. Once sent, following packets must be sent in an assigned slot, even if originating from a base station.

	<b>Description</b>	<b>Type</b>	<b>Bytes</b>	<b>Defined</b>	
	start string	uint8 [3]	3	">*>"	
	packet indicator	uint8	1	'C'	
	byte count	uint32	4		
begin byte count	robot id	uint32	4		begin checksum
	baud rate	int32	4	[bps]	
	slot length	int32	4	[# states]	end checksum
end byte count	checksum	uint32	4	crc32	



Table A.4: The environment packet builds the environment boundary polygon prior to high level control initialization. The importance function over the environment is assumed to be known by the robots, otherwise importance is assumed to be unity.

	<b>Description</b>	<b>Type</b>	<b>Bytes</b>	<b>Defined</b>	
	start string	uint8 [3]	3	">*>"	
	packet indicator	uint8	1	'E'	
	byte count	uint32	4		
begin byte count	$x_{\partial Q}$	int32	4	[mm]	begin checksum
	$y_{\partial Q}$	int32	4	[mm]	end checksum
end byte count	checksum	uint32	4	crc32	

Table A.5: The algorithm packet initializes high level control for target robots. Team members are determined from a prior communication packet, otherwise robots assume there is no team.

	<b>Description</b>	<b>Type</b>	<b>Bytes</b>	<b>Defined</b>	
	start string	uint8 [3]	3	">*>"	
	packet indicator	uint8	1	'A'	
	byte count	uint32	4		
begin byte count	robot id	uint32	4		calc checksum
end byte count	checksum	uint32	4	crc32	

Table A.6: The dynamics packet initializes low level control for target robots. Current waypoint is set to first valid position acquired after packet is received.

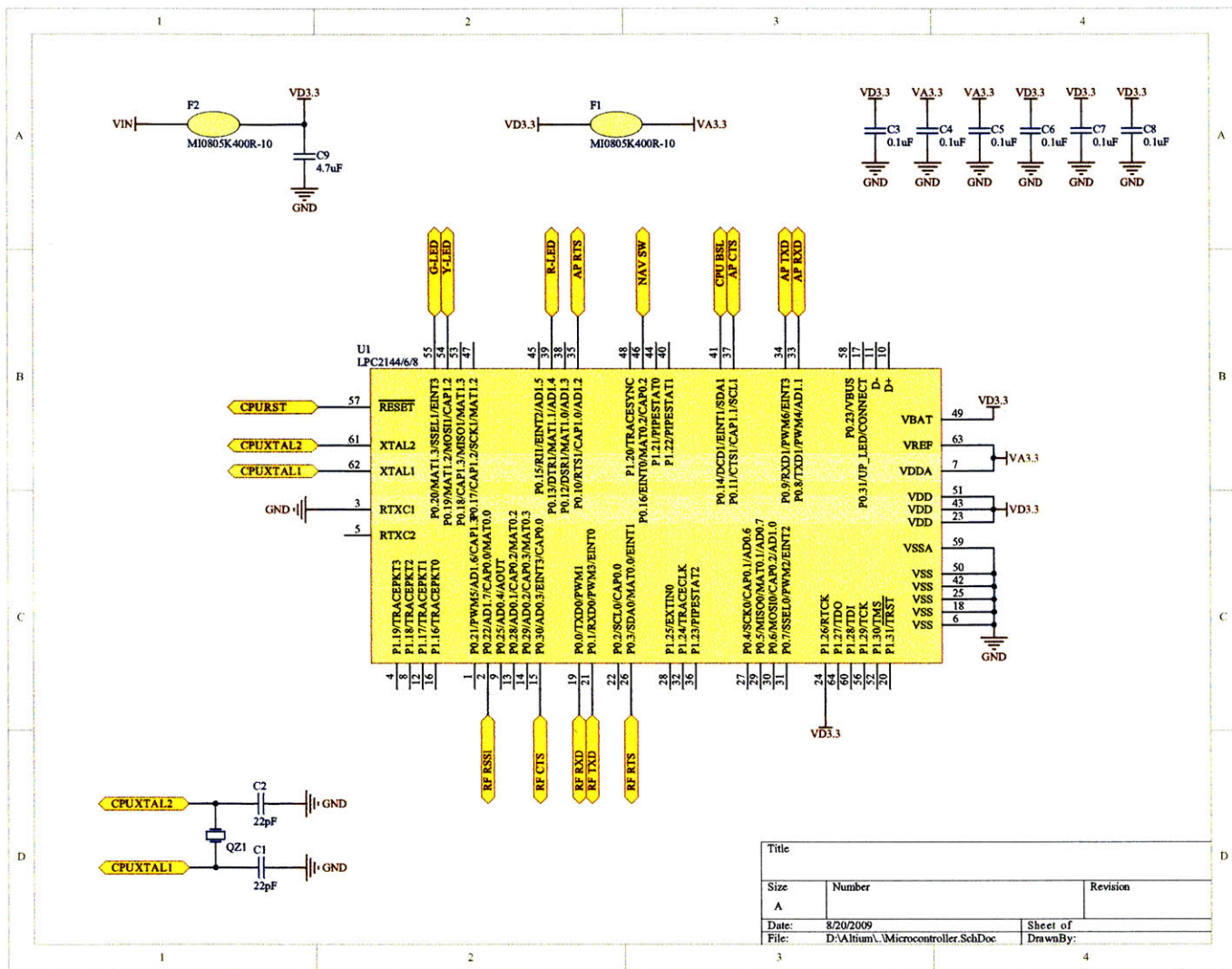
	<b>Description</b>	<b>Type</b>	<b>Bytes</b>	<b>Defined</b>	
	start string	uint8 [3]	3	">*>"	
	packet indicator	uint8	1	'D'	
	byte count	uint32	4		
begin byte count	robot id	uint32	4		calc checksum
end byte count	checksum	uint32	4	crc32	



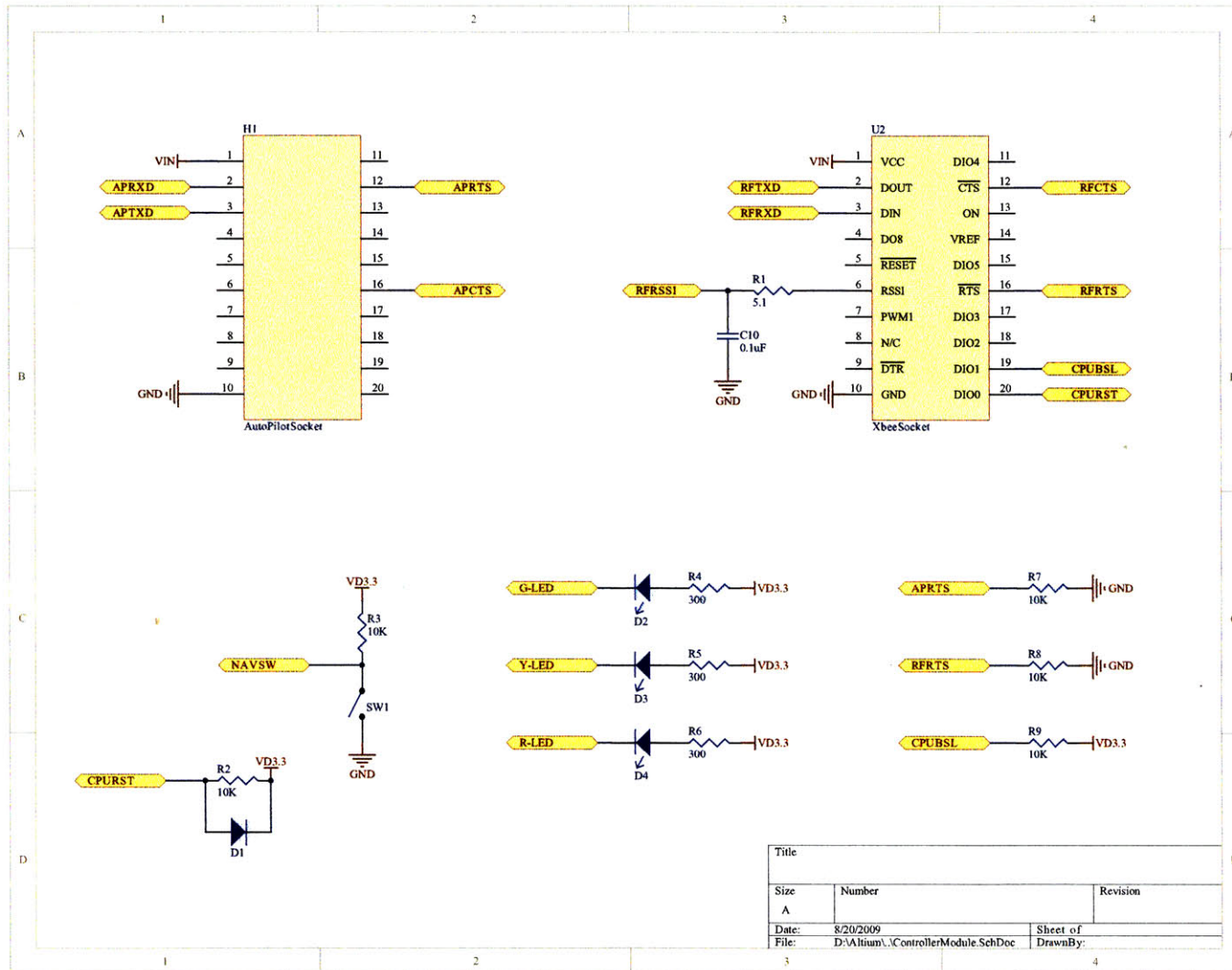
# Appendix B

## Electrical Schematics

The following figures contain the electrical schematics for the embedded control module. The first figure illustrates the installation of the NXP LPC2148 microcontroller. The second figure illustrates the port connections to the AutoPilot and radio modules, as well as the LED and navigation switch connections.



Title		
Size	Number	Revision
A		
Date:	8/20/2009	Sheet of
File:	D:\Altium\...\Microcontroller.SchDoc	DrawnBy:







# Appendix C

## Source Code

The following pages show section of the DRL quad-rotor project relevant to this thesis. The main source file, which runs the high level distributed algorithm, is first shown. All other source code is shown in alphabetical order with respect to file name. We suggest that these pages are to be used for reference only.

```

1  /*
2  *  emb/Main.h
3  *
4  *  main include for the drl quad-rotor project
5  *
6  *  Brian J. Julian
7  *
8  *  bjulian{at}mit{dot}edu
9  *
10 *  Version 0.1
11 *
12 *  31 March 2009
13 *
14 */
15
16
17 // __EMB_MAIN_H__
18
19 #ifndef __EMB_MAIN_H__
20 #define __EMB_MAIN_H__
21
22
23 // defines
24 #define MY_ROBOTNAME ((byte_t *)"alpha")
25
26
27 // includes
28
29 #include "emb/Api.h"
30 #include "emb/Alg.h"

```

```

31 #include "emb/Clock.h"
32 #include "emb/Comm.h"
33 #include "emb/Global.h"
34 #include "emb/Gps.h"
35 #include "emb/Imu.h"
36 #include "emb/Mhop.h"
37 #include "emb/Nav.h"
38 #include "emb/Status.h"
39 #include "emb/Types.h"
40 #include "lpc/Cntl.h"
41 #include "lpc/Eint.h"
42 #include "lpc/Tmr.h"
43 #include "lpc/Types.h"
44
45
46 // main struct
47
48 typedef struct
49 {
50     lpcCntl_s Cntl;
51     lpcTmr_s Tmr;
52
53     embAp_s Ap;
54     embAlg_s Alg;
55     embClock_s Clock;
56     embComm_s Comm;
57     embEnv_s Env;
58     embGlobal_s Global;
59     embGps_s Gps;
60     embImu_s Imu;

```

```
61  embMhop_s Mhop;
62  embNav_s Nav;
63  embPidList_s PidList;
64  embStateList_s StateList;
65  embStatus_s Status;
66  }
67  embMain_s;
68
69
70  // __EMB_MAIN_H__
71
72  #endif
```

5

```
1 /*
2  * emb/Main.c
3  *
4  * main source code for the drl quad-rotor project
5  *
6  * Brian J. Julian
7  *
8  * bjulian{at}mit{dot}edu
9  *
10 * Version 0.1
11 *
12 * 31 March 2009
13 *
14 */
15
16
17 #include "emb/Main.h"
18
19 #include "emb/Ap.h"
20 #include "emb/Alg.h"
21 #include "emb/Clock.h"
22 #include "emb/Comm.h"
23 #include "emb/Nav.h"
24 #include "emb/Status.h"
25 #include "lpc/Cntl.h"
26 #include "lpc/Eint.h"
27 #include "lpc/Tmr.h"
28 #include "lpc/Uart.h"
29 #include "lpc/Types.h"
30 #include "lpc/Lpc214x.h"
```

```

31
32
33 // global variables
34
35 embMain_s MallocEmb;
36
37
38 // local defines
39
40 #define HIGHLEVELMSEC.PERIOD 1000
41 #define SYS.TMR.NUM 0
42 #define RX_AP.PERIOD (CCLK/1000)
43 #define TX_AP.PERIOD (CCLK/33)
44 #define ALG.PERIOD (CCLK/1)
45
46
47 // local functions
48
49 void initEmbStruct(embMain_s *Emb);
50 void initEmb(embMain_s *Emb);
51 void configEmb(embMain_s *Emb);
52 void enableEmb(embMain_s *Emb);
53 void sysTmrVectAddr(void) __attribute__((interrupt("IRQ")));
54 void distributedAlg(embMain_s *Emb);
55
56
57
58 // main function for quadrotor project
59 int main(void)
60 {

```

```

61 // local stack
62 embMain_s *Emb;
63
64 // create pointer to memory for emb structure
65 Emb = &MallocEmb;
66
67 // start embedded system
68 initEmb(Emb);
69 configEmb(Emb);
70 enableEmb(Emb);
71
72 // infinite while loop for distributed algorithm
73 while(1)
74 {
75     embDistAlg(&Emb->Alg);
76     lpcSleepMsecTmr(&Emb->Tmr,
77                    ALG_PERIOD);
78 }
79
80 return(0);
81 }
82
83
84
85 // initialize embedded control
86 void initEmb(embMain_s *Emb)
87 {
88     // initialize lpc control structure
89     lpcInitCntl(&Emb->Cntl);
90

```

```

91 // initialize system timer
92 lpcInitTmr(&Emb->Tmr);
93
94 // assign embedded links
95 initEmbStruct(Emb);
96
97 // initialize AutoPilot
98 embInitAp(&Emb->Ap);
99
100 // initialize distributed algorithm
101 embInitAlg(&Emb->Alg);
102
103 // initialize system clock
104 embInitClock(&Emb->Clock);
105
106 // initialize comm
107 embInitComm(&Emb->Comm);
108
109 // initialize multi-hop algorithm
110 embInitEnv(&Emb->Env);
111
112 // initialize global position
113 embInitGlobal(&Emb->Global);
114
115 // initialize multi-hop algorithm
116 embInitMhop(&Emb->Mhop);
117
118 // initialize nav
119 embInitNav(&Emb->Nav);
120

```



```

121 // initialize Pid
122 embInitPid(&Emb->PidList);
123
124 // initialize state
125 embInitState(&Emb->StateList);
126
127 // initialize status
128 embInitStatus(&Emb->Status);
129 }
130
131
132
133 // config embedded control system
134 void configEmb(embMain_s *Emb)
135 {
136 // configure lpc control structure
137
138 // configure system timer
139 lpcSetNumTmr(&Emb->Tmr,
140             SYS_TMR_NUM);
141 lpcSetVectAddrTmr(&Emb->Tmr,
142                  (reg32_t)sysTmrVectAddr);
143
144 // configure AutoPilot
145 embConfigAp(&Emb->Ap);
146
147 // configure distributed algorithm
148 embConfigAlg(&Emb->Alg);
149
150 // configure system clock

```

```

151  embConfigClock(&Emb->Clock);
152
153  // configure comm
154  embConfigComm(&Emb->Comm);
155
156  // configure multi-hop algorithm
157  embConfigEnv(&Emb->Env);
158
159  // configure global position
160  embConfigGlobal(&Emb->Global);
161
162  // configure multi-hop algorithm
163  embConfigMhop(&Emb->Mhop);
164
165  // configure nav
166  embConfigNav(&Emb->Nav);
167
168  // configure Pid
169  embConfigPid(&Emb->PidList);
170
171  // configure state
172  embConfigState(&Emb->StateList);
173
174  // configure status
175  embConfigStatus(&Emb->Status);
176  }
177
178
179  // enable embedded control system
180  void enableEmb(embMain_s *Emb)

```

```
181 {
182 // enable lpc control structure
183 lpcEnableCntl(&Emb->Cntl);
184
185 // enable two system timers
186 lpcEnableTmr(&Emb->Tmr);
187
188 // enable AutoPilot
189 embEnableAp(&Emb->Ap);
190
191 // enable distributed algorithm
192 embEnableAlg(&Emb->Alg);
193
194 // enable system clock
195 embEnableClock(&Emb->Clock);
196
197 // enable comm
198 embEnableComm(&Emb->Comm);
199
200 // enable multi-hop algorithm
201 embEnableEnv(&Emb->Env);
202
203 // enable global position
204 embEnableGlobal(&Emb->Global);
205
206 // enable multi-hop algorithm
207 embEnableMhop(&Emb->Mhop);
208
209 // enable nav
210 embEnableNav(&Emb->Nav);
```

```

211
212 // enable Pid
213 embEnablePid(&Emb->PidList);
214
215 // enable state
216 embEnableState(&Emb->StateList);
217
218 // enable status
219 embEnableStatus(&Emb->Status);
220 }
221
222
223 // sets vector address for system timer
224
225 void sysTmrVectAddr(void)
226 {
227 // local stack
228 embMain_s *Emb;
229 bool_t Read;
230
231 // create linked emb
232 Emb = &MallocEmb;
233
234 // if RxComm match caused IR
235 lpcReadIrMatchTmr(&Emb->Comm.Rx.Match,
236                  &Read);
237 if(Read)
238 {
239 // rx comm handle
240 embRxComm(&Emb->Comm);

```

```

241     }
242
243     // if TxComm match caused IR
244     lpcReadIrMatchTmr(&Emb->Comm.Tx.Match,
245                      &Read);
246     if(Read)
247     {
248         // tx comm handle
249         embTxComm(&Emb->Comm);
250
251     }
252
253     // if RxAp match caused IR
254     lpcReadIrMatchTmr(&Emb->Ap.Rx.Match,
255                      &Read);
256     if(Read)
257     {
258         // tx ap handle
259         embTxAp(&Emb->Ap);
260
261     }
262
263     // if TxAp match caused IR
264     lpcReadIrMatchTmr(&Emb->Ap.Tx.Match,
265                      &Read);
266     if(Read)
267     {
268         // tx ap handle
269         embTxAp(&Emb->Ap);
270

```

```

271     }
272
273     // reset vic
274     lpcResetVicTmr(&Emb->Tmr);
275 }
276
277
278 // initialize embedded struct
279
280 void initEmbStruct(embMain_s *Emb)
281 {
282     // alg
283     Emb->Alg.Env = &Emb->Env;
284     Emb->Alg.Global = &Emb->Global;
285     Emb->Alg.StateList = &Emb->StateList;
286
287     // ap
288     Emb->Ap.Tmr = &Emb->Tmr;
289     Emb->Ap.Gps = &Emb->Gps;
290     Emb->Ap.StateList = &Emb->StateList;
291     Emb->Ap.PidList = &Emb->PidList;
292     Emb->Ap.Global = &Emb->Global;
293     Emb->Ap.Imu = &Emb->Imu;
294     Emb->Ap.Status = &Emb->Status;
295     Emb->Ap.Nav = &Emb->Nav;
296     Emb->Ap.Clock = &Emb->Clock;
297
298     // comm
299     Emb->Comm.Tmr = &Emb->Tmr;
300     Emb->Comm.Status = &Emb->Status;

```

```

301  Emb->Comm.StateList = &Emb->StateList;
302  Emb->Comm.Env = &Emb->Env;
303  Emb->Comm.PidList = &Emb->PidList;
304  Emb->Comm.Global = &Emb->Global;
305  Emb->Comm.Clock = &Emb->Clock;
306
307  // nav
308  Emb->Nav.Status = &Emb->Status;
309
310  // statelist
311  Emb->Mhop.StateList = &Emb->StateList;
312
313  // global
314  Emb->Global.StateList = &Emb->StateList;
315  Emb->Global.PidList = &Emb->PidList;
316
317  // clock
318  Emb->Clock.Tmr = &Emb->Tmr;
319
320  // assign my robot id
321  Emb->StateList.State[EMB_MY_STATE].RobotId =
        embRobotIdState(MY_ROBOT_NAME);
322  }

```

```
1 /*
2  * emb/Alg.h
3  *
4  * algorithm includes for the drl quad-rotor project
5  *
6  * Brian J. Julian
7  *
8  * bjulian{at}mit{dot}edu
9  *
10 * Version 0.1
11 *
12 * 31 March 2009
13 *
14 */
15
16
17 // __EMB_ALG_H__
18
19 #ifndef __EMB_ALG_H__
20 #define __EMB_ALG_H__
21
22
23 // includes
24
25 #include "emb/Env.h"
26 #include "emb/Global.h"
27 #include "emb/State.h"
28 #include "emb/Types.h"
29 #include "lpc/Types.h"
30
```



```

31
32 // defines
33
34 #define EMB_MAX_SIZE_ALG 10
35 #define EMB_MAX_BNDRY_SIZE_ALG 10
36 #define EMB_MAX_GRID_SIZE_ALG 10
37
38
39 // field of view struct
40
41 typedef struct
42 {
43     pos_t Pos[2];
44     int32_t Edge[2];
45 }
46     embFov_s;
47
48 // field of view list
49
50 typedef struct
51 {
52     int32_t HalfAngle[2];
53     int32_t HalfEdge[2];
54
55     embFov_s Fov[4];
56 }
57     embFovList_s;
58
59
60 // algorithm struct

```

```

61
62 typedef struct
63 {
64     int64_t PartH_PartP [4];
65     int32_t GainInvH [4];
66
67     int32_t OpticsInvA;
68     int32_t OpticsB;
69     int32_t PriorW;
70
71     int32_t Nk[4][2];
72
73     int32_t DeltaWayPosSat [4];
74     int32_t DeltaWayNegSat [4];
75
76     embFovList_s FovList [EMB_MAX_SIZE_STATE];
77
78     embEnv_s *Env;
79     embGlobal_s *Global;
80     embStateList_s *StateList;
81 }
82     embAlg_s;
83
84
85 // functions
86
87 void embInitAlg (embAlg_s *Alg);
88 void embConfigAlg (embAlg_s *Alg);
89 void embEnableAlg (embAlg_s *Alg);
90

```

```
91 void embClrAlg(embAlg_s *Alg);
92 void embDistAlg(embAlg_s *Alg);
93
94 // --EMB_ALG_H--
95
96 #endif
```

```
1 /*
2  * emb/Alg.c
3  *
4  * algorithm source code for the drl quad-rotor project
5  *
6  * Brian J. Julian
7  *
8  * bjulian{at}mit{dot}edu
9  *
10 * Version 0.1
11 *
12 * 31 March 2009
13 *
14 */
15
16
17 // includes
18
19 #include "emb/Alg.h"
20
21 #include "emb/Types.h"
22 #include "lpc/Math.h"
23 #include "lpc/Types.h"
24
25
26 // local defines
27
28 #define NUMFOV_BNDRY_SEG 10
29
30
```

```

31 // local functions
32
33 void createFovEdges(embFov_s *Fov);
34 void calcFovList(embAlg_s *Alg);
35 void calcIntegrandArea(embAlg_s *Alg);
36 void calcIntegrandEdge(embAlg_s *Alg);
37 bool_t isPntInFov(const pos_t *Pnt,
38                  const embFov_s *Fov);
39 bool_t isPntInEnv(const pos_t *Pnt,
40                  const embEnv_s *Env);
41
42 // initialize algorithm
43
44 void embInitAlg(embAlg_s *Alg)
45 { }
46
47
48 // configure algorithm
49
50 void embConfigAlg(embAlg_s *Alg)
51 { }
52
53
54 // enable algorithm
55
56 void embEnableAlg(embAlg_s *Alg)
57 { }
58
59
60 // distributed algorithm

```

```

61
62 void embDistAlg(embAlg_s *Alg)
63 {
64     // local stack
65     int32_t ii;
66     pos_t Way[4];
67
68     // clear PartH/PartP
69     for(ii = 0;
70         ii < sizeof(Alg->PartH_PartP)/sizeof(Alg->PartH_PartP
71             [0]);
72         ii++)
73     {
74         Alg->PartH_PartP[ii] = 0;
75     }
76
77     // calc new field of views based on states
78     calcFovList(Alg);
79
80     // for all grid points in environment
81     calcIntegrandArea(Alg);
82
83     // for all points on the field of view boundary
84     calcIntegrandEdge(Alg);
85
86     // update waypoints
87     for(ii = 0;
88         ii < sizeof(Way) / sizeof(Way[0]);
89         ii++)
90     {

```

```

90     Way[ ii ] = -(int32_t)(Alg->PartH_PartP [ ii ] / Alg->
        GainInvH [ ii ] );
91     Way[ ii ] = INT_MIN_MAX( Alg->DeltaWayNegSat [ ii ] ,
92                             Alg->DeltaWayPosSat [ ii ] ,
93                             Way[ ii ] );
94     Way[ ii ] += Alg->Global->Way[ ii ];
95 }
96 embUpWayGlobal( Alg->Global ,
97                Way );
98 }
99
100
101 // calc field of view list
102
103 void calcFovList( embAlg_s *Alg )
104 {
105     // local stack
106     int32_t ii ;
107     int32_t jj ;
108     int32_t kk ;
109     int32_t Cos ;
110     int32_t Sin ;
111
112     // calculate new fields of view
113     for( ii = 0 ;
114          ii < Alg->StateList->Size ;
115          ii++)
116     {
117         Cos = int32Cos( Alg->StateList->State [ ii ] . Pos [
                EMBYAWSTATE ] ) ;

```

```

118     Sin = int32Sin(Alg->StateList->State[ii].Pos[
        EMB.YAW.STATE]);
119
120     // use half angles to get edge vectors
121     for(jj = 0;
122         jj < sizeof(Alg->FovList[0].HalfAngle)/sizeof(Alg->
        FovList[0].HalfAngle[0]);
123         jj++)
124     {
125         Alg->FovList[ii].HalfEdge[jj] = int32Tan(Alg->
        FovList[ii].HalfAngle[jj]);
126         Alg->FovList[ii].HalfEdge[jj] *= Alg->StateList->
        State[ii].Pos[EMB.Z.STATE];
127         Alg->FovList[ii].HalfEdge[jj] >>= 10;;
128     }
129
130     // calculate four vertices of field of view
131
132     Alg->FovList[ii].Fov[0].Pos[0] = Cos * Alg->FovList[ii
        ].HalfEdge[0];
133     Alg->FovList[ii].Fov[0].Pos[0] += Sin * Alg->FovList[ii
        ].HalfEdge[1];
134     Alg->FovList[ii].Fov[0].Pos[0] >>= 10;
135
136     Alg->FovList[ii].Fov[0].Pos[1] = Sin * Alg->FovList[ii
        ].HalfEdge[0];
137     Alg->FovList[ii].Fov[0].Pos[1] += -Cos * Alg->FovList[
        ii].HalfEdge[1];
138     Alg->FovList[ii].Fov[0].Pos[1] >>= 10;
139

```



```

140     Alg->FovList [ ii ]. Fov [ 1 ]. Pos [ 0 ] = Cos * Alg->FovList [ ii
        ]. HalfEdge [ 0 ];
141     Alg->FovList [ ii ]. Fov [ 1 ]. Pos [ 0 ] += -Sin * Alg->FovList [
        ii ]. HalfEdge [ 1 ];
142     Alg->FovList [ ii ]. Fov [ 1 ]. Pos [ 0 ] >>= 10;
143
144     Alg->FovList [ ii ]. Fov [ 1 ]. Pos [ 1 ] = Sin * Alg->FovList [ ii
        ]. HalfEdge [ 0 ];
145     Alg->FovList [ ii ]. Fov [ 1 ]. Pos [ 1 ] += Cos * Alg->FovList [ ii
        ]. HalfEdge [ 1 ];
146     Alg->FovList [ ii ]. Fov [ 1 ]. Pos [ 1 ] >>= 10;
147
148     Alg->FovList [ ii ]. Fov [ 2 ]. Pos [ 0 ] = -Cos * Alg->FovList [ ii
        ]. HalfEdge [ 0 ];
149     Alg->FovList [ ii ]. Fov [ 2 ]. Pos [ 0 ] += -Sin * Alg->FovList [
        ii ]. HalfEdge [ 1 ];
150     Alg->FovList [ ii ]. Fov [ 2 ]. Pos [ 0 ] >>= 10;
151
152     Alg->FovList [ ii ]. Fov [ 2 ]. Pos [ 1 ] = -Sin * Alg->FovList [ ii
        ]. HalfEdge [ 0 ];
153     Alg->FovList [ ii ]. Fov [ 2 ]. Pos [ 1 ] += Cos * Alg->FovList [ ii
        ]. HalfEdge [ 1 ];
154     Alg->FovList [ ii ]. Fov [ 2 ]. Pos [ 1 ] >>= 10;
155
156     Alg->FovList [ ii ]. Fov [ 3 ]. Pos [ 0 ] = -Cos * Alg->FovList [ ii
        ]. HalfEdge [ 0 ];
157     Alg->FovList [ ii ]. Fov [ 3 ]. Pos [ 0 ] += Sin * Alg->FovList [ ii
        ]. HalfEdge [ 1 ];
158     Alg->FovList [ ii ]. Fov [ 3 ]. Pos [ 0 ] >>= 10;
159

```

```

160     Alg->FovList [ ii ].Fov [0].Pos [1] = -Sin * Alg->FovList [ ii
        ].HalfEdge [0];
161     Alg->FovList [ ii ].Fov [0].Pos [1] += -Cos * Alg->FovList [
        ii ].HalfEdge [1];
162     Alg->FovList [ ii ].Fov [0].Pos [1] >>= 10;
163
164     // create field of view edges
165     createFovEdges (Alg->FovList [ ii ].Fov);
166
167     // shift field of view under robot
168     for (jj = 0;
169         jj < sizeof (Alg->FovList [ ii ].Fov) / sizeof (Alg->
        FovList [ ii ].Fov [0]);
170         jj++)
171     {
172         for (kk = 0;
173             kk < sizeof (Alg->FovList [ ii ].Fov [0].Pos) / sizeof
        (Alg->FovList [ ii ].Fov [0].Pos [0]);
174             kk++)
175         {
176             Alg->FovList [ ii ].Fov [jj].Pos [kk] += Alg->
        StateList->State [ ii ].Pos [kk];
177         }
178     }
179 }
180 }
181
182
183 // integrand due to field of view area within environment
184

```

```

185 void calcIntegrandArea(embAlg_s *Alg)
186 {
187     // local stack
188     int32_t ii;
189     int32_t jj;
190     int64_t Integrand;
191     int64_t h_Nq_i;
192     int64_t h_Nq_j;
193
194     // for all grid points in environment
195     for(ii = 0;
196         ii < Alg->Env->GridSize;
197         ii++)
198     {
199         // if grid point is in my field of view
200         if(isPntInFov(Alg->Env->Grid[ii].Pos,
201                     Alg->FovList[EMB_MY_STATE].Fov))
202         {
203             // insert prior weight
204             h_Nq_j = Alg->PriorW;
205
206             // for all robots in team
207             for(jj = 1;
208                 jj < Alg->StateList->Size;
209                 jj++)
210             {
211                 // if grid point is in their field of view
212                 if(isPntInFov(Alg->Env->Grid[ii].Pos,
213                             Alg->FovList[jj].Fov))
214                 {

```

```

215             h_Nq_i = Alg->OpticsInvA;
216             h_Nq_i /= (Alg->OpticsB - Alg->StateList->
                        State [ jj ]. Pos [ EMB_Z_STATE ] );
217             h_Nq_i <<= 16;
218             h_Nq_i /= (Alg->OpticsB - Alg->StateList->
                        State [ jj ]. Pos [ EMB_Z_STATE ] );
219             h_Nq_j += h_Nq_i;
220         }
221     }
222
223     // intermediate calculations
224     h_Nq_i = Alg->OpticsInvA;
225     h_Nq_i /= (Alg->OpticsB - Alg->StateList->State [
                EMB_MY_STATE ]. Pos [ EMB_Z_STATE ] );
226     h_Nq_i <<= 16;
227     h_Nq_i /= (Alg->OpticsB - Alg->StateList->State [
                EMB_MY_STATE ]. Pos [ EMB_Z_STATE ] );
228     h_Nq_i += h_Nq_j;
229     h_Nq_i = (1<<16) / h_Nq_i;
230
231     // only affects Z position
232     Integrand = -2*Alg->OpticsInvA;
233     Integrand *= Alg->Env->GridSpacing;
234     Integrand *= Alg->Env->GridSpacing;
235     Integrand *= h_Nq_i;
236     Integrand *= h_Nq_i;
237     Integrand /= (Alg->OpticsB - Alg->StateList->State [
                EMB_MY_STATE ]. Pos [ EMB_Z_STATE ] );
238     Integrand /= (Alg->OpticsB - Alg->StateList->State [
                EMB_MY_STATE ]. Pos [ EMB_Z_STATE ] );

```

```

239         Integrand /= (Alg->OpticsB - Alg->StateList->State[
                EMB_MY_STATE].Pos[EMB_Z_STATE]);
240
241         // update summation of partials
242         Alg->PartH.PartP[EMB_Z_STATE] += Integrand;
243     }
244 }
245 }
246
247
248 // integrand due to field of view edge within environment
249
250 void calcIntegrandEdge(embAlg-s *Alg)
251 {
252     // local stack
253     int32_t ii;
254     int32_t jj;
255     int32_t kk;
256     int64_t h_Nq_i;
257     int64_t h_Nq_j;
258     int32_t SegLength[2];
259     int32_t Cos;
260     int32_t Sin;
261     pos_t SegPnt[2];
262     int32_t TempInt32[2];
263
264     // create segments of field of view edge
265     SegLength[0] = (Alg->FovList[EMB_MY_STATE].HalfEdge[1]>>9);
266     SegLength[0] /= NUMFOV_BNDRY_SEG;
267

```

```

268   SegLength [1] = (Alg->FovList [EMB_MY_STATE] . HalfEdge[0] >>9);
269   SegLength [1] /= NUMFOV_BNDRY_SEG;
270
271   Cos = int32Cos (Alg->StateList->State [EMB_MY_STATE] . Pos [
        EMB_YAW_STATE]);
272   Sin = int32Sin (Alg->StateList->State [EMB_MY_STATE] . Pos [
        EMB_YAW_STATE]);
273
274   // for all segments around field of view
275   for (ii = 0;
276        ii < NUMFOV_BNDRY_SEG;
277        ii++)
278   {
279       // for all field of view edges
280       for (jj = 0;
281            jj < sizeof (Alg->FovList [EMB_MY_STATE] . Fov) /
                sizeof (Alg->FovList [EMB_MY_STATE] . Fov [0]));
282            jj++)
283       {
284           // calculate coordinates of seg point
285           for (kk = 0;
286                kk < sizeof (Alg->FovList [EMB_MY_STATE] . Fov [0] .
                Edge) / sizeof (Alg->FovList [EMB_MY_STATE] .
                Fov [0] . Edge [0]));
287                kk++)
288           {
289               SegPnt [kk] = ii * Alg->FovList [EMB_MY_STATE] . Fov
                [jj] . Edge [kk];
290               SegPnt [kk] /= NUMFOV_BNDRY_SEG;

```

```

291         SegPnt [kk] += Alg->FovList [EMB_MY.STATE]. Fov [jj
           ]. Pos [kk];
292     }
293
294     // if seg point is in environment
295     if (isPntInEnv (SegPnt ,
296                   Alg->Env))
297     {
298         // insert prior weight
299         h_Nq_j = Alg->PriorW;
300
301         // for all robots in team
302         for (kk = 1;
303             kk < Alg->StateList->Size;
304             kk++)
305         {
306             // if seg point is in their field of view
307             if (isPntInFov (SegPnt ,
308                           Alg->FovList [kk]. Fov))
309             {
310                 h_Nq_i = Alg->OpticsInvA;
311                 h_Nq_i /= (Alg->OpticsB - Alg->
                           StateList->State [kk]. Pos [EMB_Z.STATE
                           ]);
312                 h_Nq_i <<= 16;
313                 h_Nq_i /= (Alg->OpticsB - Alg->
                           StateList->State [kk]. Pos [EMB_Z.STATE
                           ]);
314                 h_Nq_j += h_Nq_i;
315             }

```





```

340         // Z effect
341         TempInt32 [1] = ((INT_ABS(Alg->FovList [
          EMB_MY_STATE]. HalfAngle [0]* Alg->Nk [jj] [0])
342         +INT_ABS(Alg->FovList [
          EMB_MY_STATE]. HalfAngle [1]*
          Alg->Nk [jj] [1])) >> 10);
343         Alg->PartH_PartP [EMB_Z_STATE] += ((TempInt32
          [0]* TempInt32 [1]) >> 10);
344
345         // Yaw effect
346         TempInt32 [1] = ((SegPnt [0] - Alg->StateList->
          State [EMB_MY_STATE]. Pos [EMB_X_STATE])
347         *((-Sin*Alg->Nk [jj] [0] - Cos*Alg
          ->Nk [jj] [1]) >> 10)
348         +(SegPnt [1] - Alg->StateList->
          State [EMB_MY_STATE]. Pos [
          EMB_Y_STATE])
349         *((Cos*Alg->Nk [jj] [0] - Sin*Alg
          ->Nk [jj] [1]) >> 10));
350         Alg->PartH_PartP [EMB_YAW_STATE] -= ((TempInt32
          [0]* TempInt32 [1]) >> 10);
351     }
352 }
353 }
354
355 }
356
357
358 // create edges from field of view points
359

```

```

360 void createFovEdges(embFov_s *Fov)
361 {
362     // local stack
363     int32_t ii;
364     int32_t jj;
365
366     // for all Fov points
367     for(ii = 0;
368         ii < 4;
369         ii++)
370     {
371         for(jj = 0;
372             jj < sizeof(Fov)/sizeof(Fov[0]);
373             jj++)
374         {
375             Fov[ii].Edge[jj] = Fov[(ii+1)%4].Pos[jj];
376             Fov[ii].Edge[jj] -= Fov[ii].Pos[jj];
377         }
378     }
379 }
380
381
382
383 // is point in field of view?
384
385 bool_t isPntInFov(const pos_t *Pnt,
386                  const embFov_s *Fov)
387 {
388     // local stack
389     int32_t ii;

```

```

390  int32_t  Crossings ;
391  int32_t  T0;
392  int32_t  T1;
393
394  // zero crossings
395  Crossings = 0;
396
397  // for all field of view edges
398  for( ii = 0;
399      ii < sizeof(Fov) / sizeof(Fov[0]);
400      ii++)
401  {
402      if(Fov[ ii ].Edge[0] != 0)
403      {
404          T0 = Pnt[0]-Fov[ ii ].Pos[0];
405          T0 <<= 10;
406          T0 /= Fov[ ii ].Edge[0];
407
408          T1 = Fov[ ii ].Pos[1]-Pnt[1];
409          T1 <<= 10;
410          T1 += T0*Fov[ ii ].Edge[1];
411      }
412      else
413      {
414          T0=0;
415          T1=0;
416      }
417
418      if((T0>0) && (T0<(1<<10)) && (T1>0))
419      {

```

```

420         Crossings++;
421     }
422 }
423
424 // if odd num of crossings, return true
425 return((bool_t)((Crossings%2)==1));
426 }
427
428
429 // is point in environment?
430
431 bool_t isPntInEnv(const pos_t *Pnt,
432                 const embEnv_s *Env)
433 {
434     // local stack
435     int32_t ii;
436     int32_t Crossings;
437     int32_t T0;
438     int32_t T1;
439
440     // zero crossings
441     Crossings = 0;
442
443     // for all boundry entries
444     for(ii = 0;
445         ii < Env->BndrySize;
446         ii++)
447     {
448         if(Env->Bndry[ii].Edge[0] != 0)
449             {

```

```

450         T0 = Pnt[0] - Env->Bndry [ ii ]. Pos [ 0 ];
451         T0 <<= 10;
452         T0 /= Env->Bndry [ ii ]. Edge [ 0 ];
453
454         T1 = Env->Bndry [ ii ]. Pos [ 1 ] - Pnt [ 1 ];
455         T1 <<= 10;
456         T1 += T0*Env->Bndry [ ii ]. Edge [ 1 ];
457     }
458     else
459     {
460         T0=0;
461         T1=0;
462     }
463
464     if ((T0>0) && (T0<(1<<10)) && (T1>0))
465     {
466         Crossings++;
467     }
468 }
469
470 // if odd num of crossings, return true
471 return ((bool_t)((Crossings%2)==1));
472 }

```

```
1 /*
2  * emb/Ap.h
3  *
4  * AutoPilot includes for the drl quad-rotor project
5  *
6  * Brian J. Julian
7  *
8  * bjulian{at}mit{dot}edu
9  *
10 * Version 0.1
11 *
12 * 31 March 2009
13 *
14 */
15
16
17 // __EMB_AP_H__
18
19 #ifndef __EMB_AP_H__
20 #define __EMB_AP_H__
21
22
23 // includes
24
25 #include "emb/Clock.h"
26 #include "emb/Env.h"
27 #include "emb/Gps.h"
28 #include "emb/Global.h"
29 #include "emb/Imu.h"
30 #include "emb/Mhop.h"
```

```

31 #include "emb/Nav.h"
32 #include "emb/Pid.h"
33 #include "emb/State.h"
34 #include "emb/Status.h"
35 #include "lpc/Types.h"
36 #include "lpc/Uart.h"
37 #include "lpc/Tmr.h"
38
39
40 // rx ap struct
41
42 typedef struct
43 {
44     lpcMatchTmr_s Match;
45
46     int32_t ParseIndex;
47     uint32_t ByteCnt;
48     crc16_t CalcCrc;
49     crc16_t ReadCrc;
50     byte_t PktId;
51 }
52     embRxAp_s;
53
54
55 // tx ap struct
56
57 typedef struct
58 {
59     lpcMatchTmr_s Match;
60

```

```

61  crc16_t  Crc;
62  uint32_t ByteCnt;
63
64  lpcBuf_s  Buf;
65  byte_t   BufMem[LPC_TX_BUF_SIZE_UART];
66  }
67  embTxAp_s;
68
69
70  // ap struct
71
72  typedef struct
73  {
74    lpcTmr_s *Tmr;
75
76    lpcUart_s Uart;
77
78    embRxAp_s Rx;
79    embTxAp_s Tx;
80
81    embGps_s *Gps;
82    embNav_s *Nav;
83    embNav_s *Clock;
84    embImu_s *Imu;
85    embPidList_s *PidList;
86    embStateList_s *StateList;
87    embGlobal_s *Global;
88    embStatus_s *Status;
89  }
90  embAp_s;

```



```
91
92
93 // functions
94
95 void embInitAp(embAp_s *Ap);
96 void embConfigAp(embAp_s *Ap);
97 void embEnableAp(embAp_s *Ap);
98
99 void embInitRxAp(embAp_s *Ap);
100 void embConfigRxAp(embAp_s *Ap);
101 void embEnableRxAp(embAp_s *Ap);
102
103 void embInitTxAp(embAp_s *Ap);
104 void embConfigTxAp(embAp_s *Ap);
105 void embEnableTxAp(embAp_s *Ap);
106
107 void embRxAp(embAp_s *Ap);
108 void embTxAp(embAp_s *Ap);
109
110
111 // __EMB_AP_H__
112
113 #endif
```

```
1 /*
2  * emb/Ap.c
3  *
4  * AutoPilot source code for the drl quad-rotor project
5  *
6  * Brian J. Julian
7  *
8  * bjulian{at}mit{dot}edu
9  *
10 * Version 0.1
11 *
12 * 31 March 2009
13 *
14 */
15
16
17 // includes
18
19 #include "emb/Ap.h"
20 #include "lpc/Cntl.h"
21 #include "lpc/Tmr.h"
22 #include "lpc/Types.h"
23 #include "lpc/Uart.h"
24
25
26 // local defines
27
28 #define AP_UART_NUM 1
29 #define AP_UART_BAUD_RATE 57600
30 #define AP_UART_FLOW_CNTL LPC_FLOW_CNTL_ON_UART
```

```

31
32
33 // local functions
34
35 void builtApStruct(embAp_s *Ap);
36 bool_t isUnreadByteInUart(embAp_s *Ap);
37
38
39 // initialize AutoPilot
40
41 void embInitAp(embAp_s *Ap)
42 {
43     // initialize uart
44     lpcInitUart(&Ap->Uart);
45
46     // initialize rx ap
47     embInitRxAp(Ap);
48
49     // initialize tx ap
50     embInitTxAp(Ap);
51
52     // built ap struct
53     builtApStruct(Ap);
54 }
55
56
57 // configure AutoPilot
58
59 void embConfigAp(embAp_s *Ap)
60 {

```

```

61  // configure uart
62  lpcSetNumUart(&Ap->Uart ,
63              AP_UART_NUM);
64
65  lpcSetBaudRateUart(&Ap->Uart ,
66                   AP_UART_BAUD_RATE);
67
68  lpcSetFlowCntlUart(&Ap->Uart ,
69                   AP_UART_FLOW_CNTL);
70
71  // configure rx ap
72  embConfigRxAp(Ap);
73
74  // configure tx ap
75  embConfigTxAp(Ap);
76 }
77
78
79 // enable AutoPilot
80
81 void embEnableAp(embAp_s *Ap)
82 {
83  // enable uart
84  lpcEnableUart(&Ap->Uart);
85
86  // enable Tx ap
87  embEnableRxAp(Ap);
88
89  // enable Tx ap
90  embEnableTxAp(Ap);

```

```
91 }
92
93
94 // builds default ap struct
95
96 void builtApStruct(embAp_s *Ap)
97 { }
```

```
1 /*
2  * emb/Clock.h
3  *
4  * clock includes for the drl quad-rotor project
5  *
6  * Brian J. Julian
7  *
8  * bjulian{at}mit{dot}edu
9  *
10 * Version 0.1
11 *
12 * 31 March 2009
13 *
14 */
15
16
17 // __EMB_CLOCK_H__
18
19 #ifndef __EMB_CLOCK_H__
20 #define __EMB_CLOCK_H__
21
22
23 // includes
24
25 #include "emb/Types.h"
26 #include "lpc/Tmr.h"
27 #include "lpc/Types.h"
28
29
30 // clock struct
```

```

31
32 typedef struct
33 {
34     reg32_t Tc;
35     clock_t Base;
36
37     bool_t OverFlowChk;
38     int32_t Accum;
39
40     lpcTmr_s *Tmr;
41 }
42     embClock_s;
43
44
45 // functions
46
47 void embInitClock(embClock_s *Clock);
48 void embConfigClock(embClock_s *Clock);
49 void embEnableClock(embClock_s *Clock);
50
51 clock_t embGetClock(embClock_s *Clock);
52 void embSyncClock(embClock_s *Clock ,
53                   clock_t Tic ,
54                   int32_t InvWeight );
55
56
57 // --EMB_CLOCK_H--
58
59 #endif

```

```
1 /*
2  * emb/Clock.c
3  *
4  * clock source code for the drl quad-rotor project
5  *
6  * Brian J. Julian
7  *
8  * bjulian{at}mit{dot}edu
9  *
10 * Version 0.1
11 *
12 * 31 March 2009
13 *
14 */
15
16
17 // includes
18
19 #include "emb/Clock.h"
20 #include "emb/Types.h"
21 #include "lpc/Cntl.h"
22 #include "lpc/Tmr.h"
23 #include "lpc/Types.h"
24
25
26 // local defines
27
28 #define CCLK_DIV 59
29 #define DEF_BASE 0
30 #define CLOCK_OVERFLOW (1<<31)
```



```

31 #define OVERFLOW_VALUE 0xFFFFFFFF
32
33
34 // local functions
35
36 void overFlowChk(embClock_s *Clock);
37
38
39 // initialize clock
40
41 void embInitClock(embClock_s *Clock)
42 {
43     Clock->Base = DEF_BASE;
44 }
45
46
47 // configure clock
48
49 void embConfigClock(embClock_s *Clock)
50 { }
51
52
53 // enable clock
54
55 void embEnableClock(embClock_s *Clock)
56 { }
57
58
59 // get curr clock
60

```

```

61 clock_t embGetClock(embClock_s *Clock)
62 {
63     // get tc
64     lpcGetTcTmr(Clock->Tmr,
65                &Clock->Tc);
66
67     // check for overflow
68     overFlowChk(Clock);
69
70     // return clock
71     return(Clock->Base + (Clock->Tc / CCLK_DIV));
72 }
73
74
75 // sync clock with received input
76
77 void embSyncClock(embClock_s *Clock,
78                  clock_t Tic,
79                  int32_t InvWeight)
80 {
81     // local stack
82     int32_t Drift;
83
84     // get drift
85     Drift = (int32_t)(Tic - embGetClock(Clock));
86     Drift /= InvWeight;
87
88     // adjust base and accumulator
89     Clock->Base += Drift;
90     Clock->Accum += Drift;

```

```

91 }
92
93
94 // check for tc overflow
95
96 void overFlowChk(embClock_s *Clock)
97 {
98     // if overflow possible
99     if(Clock->Tc < CLOCK.OVER.FLOW)
100     {
101         if(Clock->OverFlowChk)
102         {
103             Clock->Base += (OVER.FLOW.VALUE / CCLK.DIV);
104             Clock->OverFlowChk = false;
105         }
106     }
107
108     // if not
109     else
110     {
111         Clock->OverFlowChk = true;
112     }
113 }

```

```
1 /*
2  * emb/Comm.h
3  *
4  * communications includes for the drl quad-rotor project
5  *
6  * Brian J. Julian
7  *
8  * bjulian{at}mit{dot}edu
9  *
10 * Version 0.1
11 *
12 * 31 March 2009
13 *
14 */
15
16
17 // __EMB_COMM_H__
18
19 #ifndef __EMB_COMM_H__
20 #define __EMB_COMM_H__
21
22
23 // includes
24
25 #include "emb/Clock.h"
26 #include "emb/Env.h"
27 #include "emb/Global.h"
28 #include "emb/Mhop.h"
29 #include "emb/Nav.h"
30 #include "emb/Pid.h"
```

```

31 #include "emb/State.h"
32 #include "emb/Status.h"
33 #include "lpc/Types.h"
34 #include "lpc/Uart.h"
35 #include "lpc/Tmr.h"
36
37
38 // rx comm struct
39
40 typedef struct
41 {
42     lpcMatchTmr_s Match;
43
44     int32_t ParseIndex;
45     uint32_t ByteCnt;
46     crc32_t CalcCrc;
47     crc32_t ReadCrc;
48     byte_t PktId;
49 }
50     embRxComm_s;
51
52
53 // tx comm struct
54
55 typedef struct
56 {
57     lpcMatchTmr_s Match;
58
59     crc32_t Crc;
60     uint32_t ByteCnt;

```

```

61
62   lpcBuf_s Buf;
63   byte_t BufMem[LPC_TX_BUF_SIZE_UART];
64 }
65   embTxComm_s;
66
67
68 // comm struct
69
70 typedef struct
71 {
72   lpcTmr_s *Tmr;
73
74   lpcUart_s Uart;
75
76   embRxComm_s Rx;
77   embTxComm_s Tx;
78
79   embClock_s *Clock;
80   embEnv_s *Env;
81   embGlobal_s *Global;
82   embMhop_s *Mhop;
83   embNav_s *Nav;
84   embPidList_s *PidList;
85   embStateList_s *StateList;
86   embStatus_s *Status;
87
88   bool_t InitLlCntl;
89   bool_t InitHlCntl;
90   bool_t InitLand;

```

```

91     bool_t InitHome;
92     bool_t InitEnv;
93 }
94     embComm_s;
95
96
97 // functions
98
99 void embInitComm(embComm_s *Comm);
100 void embConfigComm(embComm_s *Comm);
101 void embEnableComm(embComm_s *Comm);
102
103 void embInitRxComm(embComm_s *Comm);
104 void embConfigRxComm(embComm_s *Comm);
105 void embEnableRxComm(embComm_s *Comm);
106
107 void embInitTxComm(embComm_s *Comm);
108 void embConfigTxComm(embComm_s *Comm);
109 void embEnableTxComm(embComm_s *Comm);
110
111 void embRxComm(embComm_s *Comm);
112 void embTxComm(embComm_s *Comm);
113
114
115 // __EMB_COMM_H__
116
117 #endif

```

```
1  /*
2  *  emb/Comm.c
3  *
4  *  communications source code for the drl quad-rotor project
5  *
6  *  Brian J. Julian
7  *
8  *  bjulian{at}mit{dot}edu
9  *
10 *  Version 0.1
11 *
12 *  31 March 2009
13 *
14 */
15
16
17 // includes
18
19 #include "emb/Comm.h"
20 #include "lpc/Cntl.h"
21 #include "lpc/Tmr.h"
22 #include "lpc/Types.h"
23 #include "lpc/Uart.h"
24
25
26 // local defines
27
28 #define COMMUARTNUM 0
29 #define COMMUARTBAUDRATE 57600
30 #define COMMUARTFLOW_CNTL LPC_FLOW_CNTL_ON_UART
```



```

31
32
33 // local functions
34
35 void builtCommStruct(embComm_s *Comm);
36 bool_t isUnreadByteInUart(embComm_s *Comm);
37
38
39 // initialize comm
40
41 void embInitComm(embComm_s *Comm)
42 {
43     // initialize uart
44     lpcInitUart(&Comm->Uart);
45
46     // initialize rx comm
47     embInitRxComm(Comm);
48
49     // initialize tx comm
50     embInitTxComm(Comm);
51
52     // built comm struct
53     builtCommStruct(Comm);
54 }
55
56
57 // configure comm
58
59 void embConfigComm(embComm_s *Comm)
60 {

```

```

61  // configure uart
62  lpcSetNumUart(&Comm->Uart ,
63              COMMUARTNUM) ;
64
65  lpcSetBaudRateUart(&Comm->Uart ,
66                   COMMUART.BAUD_RATE) ;
67
68  lpcSetFlowCntlUart(&Comm->Uart ,
69                   COMMUART.FLOW_CNIL) ;
70
71  // configure rx comm
72  embConfigRxComm(Comm) ;
73
74  // configure tx comm
75  embConfigTxComm(Comm) ;
76 }
77
78
79 // enable comm
80
81 void embEnableComm(embComm_s *Comm)
82 {
83     // enable uart
84     lpcEnableUart(&Comm->Uart) ;
85
86     // enable Tx comm
87     embEnableRxComm(Comm) ;
88
89     // enable Tx comm
90     embEnableTxComm(Comm) ;

```

```
91 }
92
93
94 // builds default comm struct
95
96 void builtCommStruct(embComms *Comm)
97 {
98     Comm->InitLlCntl = false;
99     Comm->InitHlCntl = false;
100    Comm->InitLand = false;
101    Comm->InitHome = false;
102    Comm->InitEnv = false;
103 }
```

```
1 /*
2  * emb/Crc16.h
3  *
4  * crc16 includes for the drl quad-rotor project
5  *
6  * Brian J. Julian
7  *
8  * bjulian{at}mit{dot}edu
9  *
10 * Version 0.1
11 *
12 * 31 March 2009
13 *
14 */
15
16
17 // __EMB_CRC16_H__
18
19 #ifndef __EMB_CRC16_H__
20 #define __EMB_CRC16_H__
21
22
23 // includes
24
25 #include "emb/Types.h"
26 #include "lpc/Types.h"
27
28
29 // defines
30
```

```
31 #define EMB_CLR_CRC16 0xFF
32
33
34 // functions
35
36 void embUpCrc16(crc16_t *Crc,
37                byte_t Byte);
38
39
40 // __EMB_CRC16_H__
41
42 #endif
```

```

1 /*
2  * emb/Crc16.c
3  *
4  * crc16 source code for the drl quad-rotor project
5  *
6  * Brian J. Julian
7  *
8  * bjulian{at}mit{dot}edu
9  *
10 * Version 0.1
11 *
12 * 31 March 2009
13 *
14 */
15
16
17 // includes
18
19 #include "emb/Crc16.h"
20 #include "emb/Types.h"
21 #include "lpc/Types.h"
22
23
24 // updates crc16
25
26 void embUpCrc16(crc16_t *Crc,
27                byte_t Byte)
28 {
29     // update
30     Byte ^= ((*Crc) & 0xFF);

```

```
31  Byte ^= (Byte << 4);
32
33  *Crc >>= 8;
34  *Crc &= 0xFF;
35  *Crc |= (((crc16_t)Byte) << 8);
36  *Crc ^= (((byte_t)(Byte)) >> 4);
37  *Crc ^= (((crc16_t)(Byte)) << 3);
38 }
```

```
1 /*
2  * emb/Crc32.h
3  *
4  * crc32 includes for the drl quad-rotor project
5  *
6  * Brian J. Julian
7  *
8  * bjulian{at}mit{dot}edu
9  *
10 * Version 0.1
11 *
12 * 31 March 2009
13 *
14 */
15
16
17 // __EMB_CRC32_H__
18
19 #ifndef __EMB_CRC32_H__
20 #define __EMB_CRC32_H__
21
22
23 // includes
24
25 #include "emb/Types.h"
26 #include "lpc/Types.h"
27
28
29 // defines
30
```



```
31 #define EMB_CLR_CRC32 0x00000000
32
33
34 // functions
35
36 void embUpCrc32(crc32_t *Crc,
37                byte_t Byte);
38
39
40 // __EMB_CRC32_H__
41
42 #endif
```

```
1 /*
2  * emb/Crc32.c
3  *
4  * crc32 source code for the drl quad-rotor project
5  *
6  * Brian J. Julian
7  *
8  * bjulian{at}mit{dot}edu
9  *
10 * Version 0.1
11 *
12 * 31 March 2009
13 *
14 */
15
16
17 // includes
18
19 #include "emb/Crc32.h"
20 #include "emb/Types.h"
21 #include "lpc/Types.h"
22
23
24 // local defines
25
26 #define CRC32_POLY 0xEDB88320
27 #define CRC32_FLIP 0xFFFFFFFF
28
29
30 // updates crc32
```

```

31
32 void embUpCrc32(crc32_t *Crc,
33                 byte_t Byte)
34 {
35     // local stack
36     crc32_t Temp;
37     int32_t ii;
38
39     // update
40     *Crc ^= CRC32_FLIP;
41     Temp = (*Crc ^ Byte);
42     Temp &= 0xff;
43
44     for(ii = 0;
45         ii < 8;
46         ii++)
47     {
48         if((Temp & 1) == 1)
49         {
50             Temp >>= 1;
51             Temp ^= CRC32_POLY;
52         }
53         else
54         {
55             Temp >>= 1;
56         }
57     }
58
59     *Crc >>= 8;
60     *Crc ^= Temp;

```

```
61     *Crc ^= CRC32_FLIP;  
62 }
```

```
1 /*
2  * emb/Env.h
3  *
4  * environment includes for the drl quad-rotor project
5  *
6  * Brian J. Julian
7  *
8  * bjulian{at}mit{dot}edu
9  *
10 * Version 0.1
11 *
12 * 31 March 2009
13 *
14 */
15
16
17 // __EMB_ENV_H__
18
19 #ifndef __EMB_ENV_H__
20 #define __EMB_ENV_H__
21
22
23 // includes
24
25 #include "emb/Types.h"
26 #include "lpc/Types.h"
27
28
29 // defines
30
```

```

31 #define EMB_MAX_SIZE_ENV 10
32 #define EMB_MAX_BNDRY_SIZE_ENV 10
33 #define EMB_MAX_GRID_SIZE_ENV 10
34
35
36 // boundary struct
37
38 typedef struct
39 {
40     pos_t Pos[2];
41     int32_t Edge[2];
42 }
43     embBndry_s;
44
45
46 // grid struct
47
48 typedef struct
49 {
50     pos_t Pos[2];
51     int32_t Imp;
52 }
53     embGrid_s;
54
55
56 // environmnet struct
57
58 typedef struct
59 {
60     int32_t BndrySize;

```

```
61     embBndry_s Bndry [EMB_MAX_BNDRY_SIZE_ENV];
62
63     int32_t GridSize;
64     int32_t GridSpacing;
65     embGrid_s Grid [EMB_MAX_GRID_SIZE_ENV];
66 }
67     embEnv_s;
68
69
70 // functions
71
72 void embInitEnv (embEnv_s *Env);
73 void embConfigEnv (embEnv_s *Env);
74 void embEnableEnv (embEnv_s *Env);
75
76 void embClrEnv (embEnv_s *Env);
77
78
79 // __EMB_ENV_H__
80
81 #endif
```

```

1  /*
2  *  emb/Env.c
3  *
4  *  environment source code for the drl quad-rotor project
5  *
6  *  Brian J. Julian
7  *
8  *  bjulian{at}mit{dot}edu
9  *
10 *  Version 0.1
11 *
12 *  31 March 2009
13 *
14 */
15
16
17 // includes
18
19 #include "emb/Env.h"
20 #include "emb/Types.h"
21 #include "lpc/Types.h"
22
23
24 // initialize environment
25
26 void embInitEnv(embEnv_s *Env)
27 {
28     // clear env
29     embClrEnv(Env);
30 }

```



```

31
32
33 // configure environment
34
35 void embConfigEnv(embEnv_s *Env)
36 { }
37
38
39 // enable environment
40
41 void embEnableEnv(embEnv_s *Env)
42 { }
43
44
45 // clear environment
46
47 void embClrEnv(embEnv_s *Env)
48 {
49     // local stack
50     int32_t ii;
51     int32_t jj;
52
53     // clear sizes
54     Env->BndrySize = 0;
55     Env->GridSize = 0;
56
57     // clear bndry
58     for(ii = 0;
59         ii < EMB_MAX_BNDRY_SIZE_ENV;
60         ii++)

```

```

61     {
62         for (jj = 0;
63             jj < sizeof (Env->Bndry [0]. Pos) / sizeof (Env->Bndry
64                 [0]. Pos [0]);
65             jj++)
66             {
67                 Env->Bndry [ ii ]. Pos [ jj ] = 0;
68             }
69         for (jj = 0;
70             jj < sizeof (Env->Bndry [0]. Edge) / sizeof (Env->Bndry
71                 [0]. Edge [0]);
72             jj++)
73             {
74                 Env->Bndry [ ii ]. Edge [ jj ] = 0;
75             }
76
77     // clear grid
78     for (ii = 0;
79         ii < EMB_MAX_GRID_SIZE_ENV;
80         ii++)
81         {
82             Env->Grid [ ii ]. Imp = 0;
83
84             for (jj = 0;
85                 jj < sizeof (Env->Grid [0]. Pos) / sizeof (Env->Grid [0].
86                     Pos [0]);
87                 jj++)
88                 {

```

```

88         Env->Grid [ ii ]. Pos [ jj ] = 0;
89     }
90 }
91 }
92
93
94 // create edges from acquired bndry positions
95
96 void embCreateBndryEdgesEnv ( embEnv_s *Env )
97 {
98     // local stack
99     int32_t ii ;
100    int32_t jj ;
101
102    // for all bndry points
103    for ( ii = 0 ;
104         ii < Env->BndrySize ;
105         ii ++ )
106    {
107        for ( jj = 0 ;
108             jj < sizeof ( Env->Bndry [ 0 ]. Pos ) / sizeof ( Env->Bndry
109                 [ 0 ]. Pos [ 0 ] ) ;
110             jj ++ )
111        {
112            Env->Bndry [ ii ]. Edge [ jj ] = Env->Bndry [ ( ii + 1 ) % Env->
113                BndrySize ]. Pos [ jj ] ;
114            Env->Bndry [ ii ]. Edge [ jj ] -= Env->Bndry [ ii ]. Pos [ jj ] ;
115        }
116    }
117 }

```

```
1 /*
2  * emb/Global.h
3  *
4  * global position includes for quadrotor project
5  *
6  * Brian J. Julian
7  *
8  * bjulian{at}mit{dot}edu
9  *
10 * Version 0.1
11 *
12 * 31 March 2009
13 *
14 */
15
16
17 // __EMB_GLOBAL_H__
18
19 #ifndef __EMB_GLOBAL_H__
20 #define __EMB_GLOBAL_H__
21
22
23 // includes
24
25 #include "emb/Pid.h"
26 #include "emb/State.h"
27 #include "emb/Types.h"
28 #include "ipc/Types.h"
29
30
```

```

31 // global struct
32
33 typedef struct
34 {
35     clock_t TimeStamp;
36
37     pos_t So3[6];
38     pos_t Yaw;
39
40     pos_t Way[4];
41     pos_t OldWay[4];
42     bool_t WayActive;
43     pos_t Err[4];
44
45     int32_t R_b[9];
46
47     embPidList_s *PidList;
48     embStateList_s *StateList;
49 }
50     embGlobal_s;
51
52
53 // functions
54
55 void embInitGlobal(embGlobal_s *Global);
56 void embConfigGlobal(embGlobal_s *Global);
57 void embEnableGlobal(embGlobal_s *Global);
58
59 void embUpGlobal(embGlobal_s *Global,
60                 const embGlobal_s *NewGlobal);

```

```
61 void embUpWayGlobal(embGlobal_s *Global,  
62                     const pos_t *Way);  
63 void embUpPidFromGlobal(const embGlobal_s *Global);  
64 void embUpStateFromGlobal(const embGlobal_s *Global);  
65  
66  
67 // --EMB_GLOBAL_H--  
68  
69 #endif
```

```
1 /*
2  * emb/Global.c
3  *
4  * global pos source code for the drl quad-rotor project
5  *
6  * Brian J. Julian
7  *
8  * bjulian{at}mit{dot}edu
9  *
10 * Version 0.1
11 *
12 * 31 March 2009
13 *
14 */
15
16
17 // includes
18
19 #include "emb/Global.h"
20 #include "emb/State.h"
21 #include "emb/Types.h"
22 #include "lpc/Math.h"
23 #include "lpc/Types.h"
24
25
26 // initialize global position
27
28 void embInitGlobal(embGlobal_s *Global)
29 { }
30
```

```

31
32 // configure global position
33
34 void embConfigGlobal(embGlobal_s *Global)
35 { }
36
37
38 // enable global position
39
40 void embEnableGlobal(embGlobal_s *Global)
41 { }
42
43
44 // update waypoint position
45
46 void embUpWayGlobal(embGlobal_s *Global,
47                    const pos_t *Way)
48 {
49     // local stack
50     int32_t ii;
51
52     // update waypoint
53     for(ii = 0;
54         ii < sizeof(Global->Way)/sizeof(Global->Way[0]);
55         ii++)
56     {
57         if(Global->WayActive)
58         {
59             Global->OldWay[ii] = Global->Way[ii];
60         }

```



```

61     Global->Way[ ii ] = Way[ ii ];
62     }
63     Global->WayActive = false;
64 }
65
66
67 // update global position
68
69 void embUpGlobal(embGlobal_s *Global,
70                 const embGlobal_s *NewGlobal)
71 {
72     // local stack
73     int32_t Psi;
74     int32_t SinPsi;
75     int32_t CosPsi;
76     int32_t T_c;
77     int32_t ii;
78
79     // update time stamp
80     Global->TimeStamp = NewGlobal->TimeStamp;
81
82     // update so3
83     for( ii = 0;
84         ii < sizeof(NewGlobal->So3)/sizeof(NewGlobal->So3[0]);
85         ii++)
86     {
87         Global->So3[ ii ] = NewGlobal->So3[ ii ];
88     }
89
90     // matrix params

```

```

91  Psi  = int32Pow(Global->So3[3],
92              2);
93  Psi += int32Pow(Global->So3[4],
94              2);
95  Psi += int32Pow(Global->So3[5],
96              2);
97  Psi  = int32Sqrt(Psi);
98
99  CosPsi = int32Cos(Psi);
100 SinPsi = int32Sin(Psi);
101
102 T_c = (1<<10) - CosPsi;
103
104 // normalize so3 angles
105 for( ii = 3;
106     ii < 6;
107     ii++)
108     {
109     Global->So3[ii] <<= 10;
110     Global->So3[ii] /= Psi;
111     }
112
113 // rotational matrix
114 Global->R_b[0] = (((((T_c*Global->So3[4]*Global->So3[3]) >>
115                    10) + (Global->So3[5]*SinPsi)) >> 10);
116 Global->R_b[1] = (((((T_c*Global->So3[4]*Global->So3[4]) >>
117                    20) + CosPsi));
118 Global->R_b[2] = (((((T_c*Global->So3[4]*Global->So3[5]) >>
119                    10) - (Global->So3[3]*SinPsi)) >> 10);

```

```

118 Global->R_b[3] = (((T_c*Global->So3[3]*Global->So3[3]) >>
    20) + CosPsi));
119 Global->R_b[4] = (((T_c*Global->So3[3]*Global->So3[4]) >>
    10) - (Global->So3[5]*SinPsi)) >> 10);
120 Global->R_b[5] = (((T_c*Global->So3[3]*Global->So3[5]) >>
    10) + (Global->So3[4]*SinPsi)) >> 10);
121
122 Global->R_b[6] = -((((T_c*Global->So3[5]*Global->So3[3]) >>
    10) - (Global->So3[4]*SinPsi)) >> 10);
123 Global->R_b[7] = -((((T_c*Global->So3[5]*Global->So3[4]) >>
    10) + (Global->So3[3]*SinPsi)) >> 10);
124 Global->R_b[8] = -((((T_c*Global->So3[5]*Global->So3[5]) >>
    20) + CosPsi));
125
126 // update yaw
127 Global->Yaw = int32Atan2(-Global->R_b[0],
128                        Global->R_b[3]);
129
130 // update way error
131 for(ii = 0;
132     ii < 3;
133     ii++)
134     {
135     Global->Err[ii] = Global->Way[ii] - Global->So3[ii];
136     }
137 Global->Err[3] = Global->Way[3] - Global->Yaw;
138 }
139
140
141

```

```

142 void embUpStateFromGlobal(const embGlobal_s *Global)
143 {
144     // local stack
145     int32_t ii;
146
147     // update my state and prepare body coord
148     for(ii = 0;
149         ii < 3;
150         ii++)
151     {
152         Global->StateList->State[EMB_MY_STATE].Pos[ii] = Global
            ->So3[ii];
153     }
154     Global->StateList->State[EMB_MY_STATE].Pos[3] = Global->Yaw
        ;
155 }
156
157
158 void embUpPidFromGlobal(const embGlobal_s *Global)
159 {
160     // local stack
161     int32_t ii;
162     int32_t jj;
163
164     // update my body coords
165     for(ii = 0;
166         ii < 3;
167         ii++)
168     {
169         // store prev coords

```

```

170     Global->PidList->Pid [ ii ]. PosB [ 1 ] = Global->PidList->Pid
        [ ii ]. PosB [ 0 ];
171     Global->PidList->Pid [ ii ]. ErrB [ 1 ] = Global->PidList->Pid
        [ ii ]. ErrB [ 0 ];
172
173     // zero curr coords
174     Global->PidList->Pid [ ii ]. PosB [ 0 ] = 0;
175     Global->PidList->Pid [ ii ]. ErrB [ 0 ] = 0;
176
177     // calc curr coords
178     for ( jj = 0;
179         jj < 3;
180         jj++)
181     {
182         Global->PidList->Pid [ ii ]. PosB [ 0 ] += Global->R_b [ 3 *
            ii + jj ] * Global->So3 [ jj ];
183         Global->PidList->Pid [ ii ]. ErrB [ 0 ] += Global->R_b [ 3 *
            ii + jj ] * Global->Err [ jj ];
184     }
185
186     Global->PidList->Pid [ ii ]. PosB [ 0 ] >>= 10;
187     Global->PidList->Pid [ ii ]. ErrB [ 0 ] >>= 10;
188 }
189 Global->PidList->Pid [ 3 ]. PosB [ 0 ] = -Global->Yaw;
190 Global->PidList->Pid [ 3 ]. ErrB [ 0 ] = -Global->Err [ 3 ];
191 }

```

```
1 /*
2  * emb/Nav.h
3  *
4  * navigation includes for the drl quad-rotor project
5  *
6  * Brian J. Julian
7  *
8  * bjulian{at}mit{dot}edu
9  *
10 * Version 0.1
11 *
12 * 31 March 2009
13 *
14 */
15
16
17 // __EMB_NAV_H__
18
19 #ifndef __EMB_NAV_H__
20 #define __EMB_NAV_H__
21
22
23 // includes
24
25 #include "emb/Status.h"
26 #include "emb/Types.h"
27 #include "lpc/Eint.h"
28 #include "lpc/Types.h"
29
30
```

```

31 // navigation flight enum
32
33 typedef enum
34 {
35     NAV_Indoor = true ,
36     NAV_Outdoor = false
37 }
38     embFlight_e;
39
40
41 // nav struct
42
43 typedef struct
44 {
45     lpcEint_s Eint;
46     embFlight_e Flight;
47
48     embStatus_s *Status;
49 }
50     embNav_s;
51
52
53 // functions
54
55 void embInitNav(embNav_s *Nav);
56 void embConfigNav(embNav_s *Nav);
57 void embEnableNav(embNav_s *Nav);
58
59
60 // __EMB_NAV_H__

```

61

62 **#endif**



```
1 /*
2  * emb/Nav.c
3  *
4  * navigation source code for the drl quad-rotor project
5  *
6  * Brian J. Julian
7  *
8  * bjulian{at}mit{dot}edu
9  *
10 * Version 0.1
11 *
12 * 31 March 2009
13 *
14 */
15
16
17 // includes
18
19 #include "emb/Nav.h"
20 #include "emb/Status.h"
21 #include "emb/Types.h"
22 #include "lpc/Eint.h"
23 #include "lpc/Types.h"
24
25
26 // local defines
27
28 #define NAV_EINT_PIN 7
29
30
```

```

31 // global pointer
32
33 embNav_s *LinkedNav;
34
35
36 // local functions
37
38 void initNavStruct(embNav_s *Nav);
39 void navEintVectAddr(void) __attribute__((interrupt("IRQ")))
    ;
40
41
42 // initialize navigation
43
44 void embInitNav(embNav_s *Nav)
45 {
46     // create internal link for interrupt
47     LinkedNav = Nav;
48
49     // initialize external interrupts
50     lpcInitEint(&Nav->Eint);
51 }
52
53
54 // configure navigation
55
56 void embConfigNav(embNav_s *Nav)
57 {
58     // configure navigation external interrupt
59     lpcSetPinEint(&Nav->Eint ,

```

```

60             NAV_EINT_PIN);
61     lpcSetVectAddrEint(&Nav->Eint,
62             (reg32_t)navEintVectAddr);
63 }
64
65
66 // enable navigation
67
68 void embEnableNav(embNav_s *Nav)
69 {
70     // enable external interrupts
71     lpcEnableEint(&Nav->Eint);
72 }
73
74
75 // eint handle
76
77 void navEintVectAddr(void)
78 {
79     // local stack
80     embNav_s *Nav;
81
82     // assigned linked nav
83     Nav = LinkedNav;
84
85     // change nav state
86     lpcReadPinEint(&Nav->Eint,
87             (bool_t *)&Nav->Flight);
88
89     switch(Nav->Flight)

```

```

90     {
91     case(NAV_Indoor):
92         Nav->Status->Green.Speed = LED_Fast;
93         break;
94     case(NAV_Outdoor):
95         Nav->Status->Green.Speed = LED_Slow;
96         break;
97     default:
98         break;
99     }
100
101     // clear interrupt request
102     lpcClearIrEint(&Nav->Eint);
103
104     // reset vic
105     lpcResetVicEint(&Nav->Eint);
106 }

```

```
1 /*
2  * emb/RxAp.c
3  *
4  * rx apunications source code for the drl quad-rotor project
5  *
6  * Brian J. Julian
7  *
8  * bjulian{at}mit{dot}edu
9  *
10 * Version 0.1
11 *
12 * 31 March 2009
13 *
14 */
15
16
17 // includes
18
19 #include "emb/Clock.h"
20 #include "emb/Ap.h"
21 #include "emb/Crc16.h"
22 #include "emb/Gps.h"
23 #include "emb/Imu.h"
24 #include "emb/State.h"
25 #include "emb/Status.h"
26 #include "emb/Types.h"
27 #include "lpc/Cntl.h"
28 #include "lpc/Math.h"
29 #include "lpc/Types.h"
30 #include "lpc/Uart.h"
```

```

31
32
33 // local defines
34
35 #define RX_AP_PERIOD (CCLK/10000)
36 #define RX_MATCHSLOT 2
37
38 #define CRC_RST EMB_CLR_CRC16
39
40 #define AP_RANGE 30000
41
42 #define START_STR ">*>"
43 #define BYTE_CNT_MIN 0
44 #define BYTE_CNT_MAX 100
45 #define STATE_SENDER_INDEX 0
46 #define RST_PARSE_INDEX 0
47
48 #define GPS_PKT_ID 0x29
49 #define IMU_PKT_ID 0x03
50
51 #define START_STR_PARSE_INDEX 1
52 #define BYTE_CNT_PARSE_INDEX (START_STR_PARSE_INDEX+1)
53 #define PKT_ID_PARSE_INDEX (BYTE_CNT_PARSE_INDEX+sizeof(
    int16_t))
54 #define GPS_PKT_PARSE_INDEX 1000
55 #define IMU_PKT_PARSE_INDEX 2000
56
57
58 // local functions
59

```

```

60 void initRxStructAp(embAp_s *Ap);
61 void discardApCrc(embAp_s *Ap);
62 bool_t isApCrcValid(embAp_s *Ap);
63 bool_t isInApRange(embAp_s *Ap,
64                    embState_s *State);
65 bool_t isUnreadByteInApUart(embAp_s *Ap);
66 void parseApByteCnt(embAp_s *Ap);
67 void parseApGpsPkt(embAp_s *Ap);
68 void parseApImuPkt(embAp_s *Ap);
69 void parseApPktId(embAp_s *Ap);
70 void parseApStartStr(embAp_s *Ap);
71
72
73 // init rx ap
74
75 void embInitRxAp(embAp_s *Ap)
76 {
77     // build rx structure
78     initRxStructAp(Ap);
79
80     // initialize sys match timer
81     lpeInitMatchTmr(&Ap->Rx.Match,
82                   Ap->Tmr);
83 }
84
85
86 // config rx ap
87
88 void embConfigRxAp(embAp_s *Ap)
89 {

```

```

90  // configure rx match
91  lpcSetSlotMatchTmr(&Ap->Rx.Match,
92                    RX_MATCH_SLOT);
93  }
94
95
96  // enable rx ap
97
98  void embEnableRxAp(embAp_s *Ap)
99  {
100  // enable rx match
101  lpcEnableMatchTmr(&Ap->Rx.Match);
102
103  // reset rx match
104  lpcResetMatchTmr(&Ap->Rx.Match,
105                  CCLK);
106  }
107
108
109  // receive ap handle
110
111  void embRxAp(embAp_s *Ap)
112  {
113  // while unread bytes in buf
114  while(isUnreadByteInApUart(Ap))
115  {
116  // update parse index and figure out which case
117  switch(++(Ap->Rx.ParseIndex))
118  {
119  // if start string index

```



```

120     case (START_STR_PARSEINDEX) :
121         parseApStartStr (Ap) ;
122         break ;
123
124         // if byte count index
125     case (BYTE_CNT_PARSEINDEX) :
126         parseApByteCnt (Ap) ;
127         break ;
128
129         // if packet id index
130     case (PKT_ID_PARSEINDEX) :
131         parseApPktId (Ap) ;
132         break ;
133
134         // if environment packet index
135     case (GPS_PKT_PARSEINDEX) :
136         parseApGpsPkt (Ap) ;
137         break ;
138
139         // if homing index
140     case (IMU_PKT_PARSEINDEX) :
141         parseApImuPkt (Ap) ;
142         break ;
143     }
144 }
145
146 // reset rx ap match
147 lpcResetMatchTmr (&Ap->Rx.Match ,
148                  RX_AP_PERIOD) ;
149

```

```

150  // clear rx ap match IR
151  lpcClrIrMatchTmr(&Ap->Rx.Match);
152  }
153
154
155  // builds rx ap struct
156
157  void initRxStructAp(embAp_s *Ap)
158  { }
159
160
161  // checks if calculated crc is same as read one
162
163  bool_t isApCrcValid(embAp_s *Ap)
164  {
165      // local stack
166      int32_t ii;
167      byte_t ReadByte;
168      bool_t DummyBool;
169      int32_t BytesRecv;
170
171      // unread packet for crc
172      lpcOffsetReadUart(&Ap->Uart,
173                      -Ap->Rx.ByteCnt);
174
175      // reset crcs
176      Ap->Rx.CalcCrc = CRC_RST;
177      Ap->Rx.ReadCrc = CRC_RST;
178
179      // calculate crc of packet

```

```

180  for( ii = 0;
181      ii < Ap->Rx.ByteCnt - sizeof(Ap->Rx.CalcCrc);
182      ii++)
183  {
184      lpcReadByteUart(&Ap->Uart ,
185                    &ReadByte ,
186                    &DummyBool);
187      embUpCrc16(&Ap->Rx.CalcCrc ,
188                ReadByte);
189  }
190
191  // read crc from packet
192  LPC_READ_UART(Ap->Uart ,
193                Ap->Rx.ReadCrc ,
194                BytesRecv);
195
196  // return match comparison
197  return((bool_t)(Ap->Rx.CalcCrc == Ap->Rx.ReadCrc));
198 }
199
200
201 // check for valid start string at beginning of packet
202
203 void parseApStartStr(embAp_s *Ap)
204 {
205     // local stack
206     int32_t ii;
207     bool_t ValidStartStr;
208     bool_t DummyBool;
209     byte_t ReadByte;

```

```

210
211 // unread bytes for start string comparison
212 lpcOffsetReadUart(&Ap->Uart ,
213                 -(sizeof(START_STR)-1));
214
215 // verify start string
216 for(ii = 0, ValidStartStr = true;
217     ii < sizeof(START_STR)-1;
218     ii++)
219 {
220     lpcReadByteUart(&Ap->Uart ,
221                  &ReadByte ,
222                  &DummyBool);
223
224     ValidStartStr = ValidStartStr && ((bool_t)(ReadByte ==
225         START_STR[ii]));
226 }
227 // reset parse if not valid start string
228 if(!ValidStartStr)
229 {
230     Ap->Rx.ParseIndex = RST_PARSE_INDEX;
231 }
232 }
233
234
235 // parse byte count from packet
236
237 void parseApByteCnt(embAp-s *Ap)
238 {

```

```

239 // local stack
240 int32_t BytesRecv;
241
242 // read byte count from uart
243 lpcOffsetReadUart(&Ap->Uart ,
244                 -sizeof(Ap->Rx.ByteCnt));
245 LPC_READ_UART(Ap->Uart ,
246              Ap->Rx.ByteCnt ,
247              BytesRecv);
248 }
249
250
251 // parse packet id from packet
252
253 void parseApPktId(embAp_s *Ap)
254 {
255 // local stack
256 bool_t DummyBool;
257 bool_t ValidByteCnt;
258
259 // read byte count from uart
260 lpcOffsetReadUart(&Ap->Uart ,
261                 -sizeof(byte_t));
262 lpcReadByteUart(&Ap->Uart ,
263               &Ap->Rx.PktId ,
264               &DummyBool);
265
266 // check if valid byte count
267 ValidByteCnt = (bool_t)((Ap->Rx.ByteCnt >= BYTE_CNT_MIN) &&
268                      (Ap->Rx.ByteCnt <= BYTE_CNT_MAX));

```

```

269
270 // set parse index if valid byte count
271 if(ValidByteCnt)
272 {
273     // assign corresponding parse index for valid packet id
274     switch(Ap->Rx.PktId)
275     {
276         // if gps packet
277         case(GPS_PKT_ID):
278             Ap->Rx.ParseIndex = GPS_PKT_PARSEINDEX;
279             Ap->Rx.ParseIndex -= Ap->Rx.ByteCnt;
280             break;
281
282         // if imu packet
283         case(IMU_PKT_ID):
284             Ap->Rx.ParseIndex = IMU_PKT_PARSEINDEX;
285             Ap->Rx.ParseIndex -= Ap->Rx.ByteCnt;
286             break;
287
288         // reset if not valid packet
289         default:
290             Ap->Rx.ParseIndex = RST_PARSEINDEX;
291             break;
292     }
293 }
294
295 // reset parse index if not valid byte count
296 else
297 {
298     Ap->Rx.ParseIndex = RST_PARSEINDEX;

```

```

299     }
300 }
301
302
303 // parse gps packet
304
305 void parseApGpsPkt(embAp.s *Ap)
306 {
307     // local stack
308     int32_t BytesRecv;
309
310     // if valid checksum
311     if(isApCrcValid(Ap))
312     {
313         // unread packet for gps information
314         lpcOffsetReadUart(&Ap->Uart ,
315                         -Ap->Rx.ByteCnt);
316
317         // read data
318         LPC_READ_UART(Ap->Uart ,
319                     Ap->Gps->Latitude ,
320                     BytesRecv);
321
322         LPC_READ_UART(Ap->Uart ,
323                     Ap->Gps->Longitude ,
324                     BytesRecv);
325
326         LPC_READ_UART(Ap->Uart ,
327                     Ap->Gps->Height ,
328                     BytesRecv);

```

```

329
330     LPC_READ_UART(Ap->Uart ,
331                   Ap->Gps->SpeedX ,
332                   BytesRecv) ;
333
334     LPC_READ_UART(Ap->Uart ,
335                   Ap->Gps->SpeedY ,
336                   BytesRecv) ;
337
338     LPC_READ_UART(Ap->Uart ,
339                   Ap->Gps->Heading ,
340                   BytesRecv) ;
341
342     LPC_READ_UART(Ap->Uart ,
343                   Ap->Gps->HorizontalAccuracy ,
344                   BytesRecv) ;
345
346     LPC_READ_UART(Ap->Uart ,
347                   Ap->Gps->VerticalAccuracy ,
348                   BytesRecv) ;
349
350     LPC_READ_UART(Ap->Uart ,
351                   Ap->Gps->SpeedAccuracy ,
352                   BytesRecv) ;
353
354     LPC_READ_UART(Ap->Uart ,
355                   Ap->Gps->NumSV ,
356                   BytesRecv) ;
357
358     LPC_READ_UART(Ap->Uart ,

```



```

359             Ap->Gps->Status ,
360             BytesRecv);
361
362     LPC_READ_UART(Ap->Uart ,
363                 Ap->Gps->BestEstimateLatitude ,
364                 BytesRecv);
365
366     LPC_READ_UART(Ap->Uart ,
367                 Ap->Gps->BestEstimateLongitude ,
368                 BytesRecv);
369
370     LPC_READ_UART(Ap->Uart ,
371                 Ap->Gps->BestEstimateSpeedX ,
372                 BytesRecv);
373
374     LPC_READ_UART(Ap->Uart ,
375                 Ap->Gps->BestEstimateSpeedY ,
376                 BytesRecv);
377
378     // discard trailing checksum
379     discardApCrc(Ap);
380 }
381
382 // reset parse index
383 Ap->Rx.ParseIndex = RST_PARSE_INDEX;
384 }
385
386
387 // parse imu packet
388

```

```

389 void parseApImuPkt(embAp_s *Ap)
390 {
391     // local stack
392     int32_t BytesRecv;
393
394     // if valid checksum
395     if(isApCrcValid(Ap))
396     {
397         // unread packet for imu information
398         lpcOffsetReadUart(&Ap->Uart ,
399                         -Ap->Rx.ByteCnt);
400
401         // read data
402         LPC_READ_UART(Ap->Uart ,
403                     Ap->Imu->AngleNick ,
404                     BytesRecv);
405
406         LPC_READ_UART(Ap->Uart ,
407                     Ap->Imu->AngleRoll ,
408                     BytesRecv);
409
410         LPC_READ_UART(Ap->Uart ,
411                     Ap->Imu->AngleYaw ,
412                     BytesRecv);
413
414         LPC_READ_UART(Ap->Uart ,
415                     Ap->Imu->AngVelNick ,
416                     BytesRecv);
417
418         LPC_READ_UART(Ap->Uart ,

```

```
419             Ap->Imu->AngVelRoll ,
420             BytesRecv ) ;
421
422     LPC_READ_UART( Ap->Uart ,
423                 Ap->Imu->AngVelYaw ,
424                 BytesRecv ) ;
425
426     LPC_READ_UART( Ap->Uart ,
427                 Ap->Imu->CalibAccX ,
428                 BytesRecv ) ;
429
430     LPC_READ_UART( Ap->Uart ,
431                 Ap->Imu->CalibAccY ,
432                 BytesRecv ) ;
433
434     LPC_READ_UART( Ap->Uart ,
435                 Ap->Imu->CalibAccZ ,
436                 BytesRecv ) ;
437
438     LPC_READ_UART( Ap->Uart ,
439                 Ap->Imu->AccX ,
440                 BytesRecv ) ;
441
442     LPC_READ_UART( Ap->Uart ,
443                 Ap->Imu->AccY ,
444                 BytesRecv ) ;
445
446     LPC_READ_UART( Ap->Uart ,
447                 Ap->Imu->AccZ ,
448                 BytesRecv ) ;
```

```
449
450     LPC_READ_UART(Ap->Uart ,
451                   Ap->Imu->AccAngleNick ,
452                   BytesRecv ) ;
453
454     LPC_READ_UART(Ap->Uart ,
455                   Ap->Imu->AccAngleRoll ,
456                   BytesRecv ) ;
457
458     LPC_READ_UART(Ap->Uart ,
459                   Ap->Imu->AbsAcc ,
460                   BytesRecv ) ;
461
462     LPC_READ_UART(Ap->Uart ,
463                   Ap->Imu->Hx ,
464                   BytesRecv ) ;
465
466     LPC_READ_UART(Ap->Uart ,
467                   Ap->Imu->Hy ,
468                   BytesRecv ) ;
469
470     LPC_READ_UART(Ap->Uart ,
471                   Ap->Imu->Hz ,
472                   BytesRecv ) ;
473
474     LPC_READ_UART(Ap->Uart ,
475                   Ap->Imu->MagHeading ,
476                   BytesRecv ) ;
477
478     LPC_READ_UART(Ap->Uart ,
```

```

479             Ap->Imu->SpeedX ,
480             BytesRecv );
481
482     LPC_READ_UART(Ap->Uart ,
483                 Ap->Imu->SpeedY ,
484                 BytesRecv );
485
486     LPC_READ_UART(Ap->Uart ,
487                 Ap->Imu->SpeedZ ,
488                 BytesRecv );
489
490     LPC_READ_UART(Ap->Uart ,
491                 Ap->Imu->Height ,
492                 BytesRecv );
493
494     LPC_READ_UART(Ap->Uart ,
495                 Ap->Imu->dHeight ,
496                 BytesRecv );
497
498     LPC_READ_UART(Ap->Uart ,
499                 Ap->Imu->dHeightRef ,
500                 BytesRecv );
501
502     LPC_READ_UART(Ap->Uart ,
503                 Ap->Imu->HeightRef ,
504                 BytesRecv );
505
506     // discard trailing checksum
507     discardApCrc (Ap);
508 }

```

```

509
510  // reset parse index
511  Ap->Rx.ParseIndex = RST_PARSE_INDEX;
512 }
513
514
515 // discard trailing checksum
516
517 void discardApCrc(embAp_s *Ap)
518 {
519  // local stack
520  crc16_t Crc;
521  int32_t BytesRecv;
522
523  // discard
524  LPC_READ_UART(Ap->Uart,
525                Crc,
526                BytesRecv);
527 }
528
529
530 // check if in ap range
531
532 bool_t isInApRange(embAp_s *Ap,
533                    embState_s *State)
534 {
535  // local stack
536  int32_t Dist;
537
538  // calc distance to sender

```

```

539     Dist = int32Dist(Ap->StateList->State[EMB_MY_STATE].Pos,
540                     State->Pos,
541                     3);
542
543     // if in ap range, return true, else false
544     if(Dist < APRANGE)
545     {
546         return(true);
547     }
548     else
549     {
550         return(false);
551     }
552 }
553
554 // see if unread bytes still in uart
555
556 bool_t isUnreadByteInApUart(embAp_s *Ap)
557 {
558     // local stack
559     bool_t ByteRecv;
560     byte_t DummyByte;
561
562     // function
563     lpcReadByteUart(&Ap->Uart,
564                   &DummyByte,
565                   &ByteRecv);
566
567     // return bool
568     return(ByteRecv);

```

569 }



```
1 /*
2  * emb/RxComm.c
3  *
4  * rx communications source code for the drl quad-rotor
5  * project
6  *
7  * Brian J. Julian
8  *
9  * bjulian{at}mit{dot}edu
10 *
11 * Version 0.1
12 *
13 * 31 March 2009
14 */
15
16
17 // includes
18
19 #include "emb/Clock.h"
20 #include "emb/Comm.h"
21 #include "emb/Crc32.h"
22 #include "emb/State.h"
23 #include "emb/Status.h"
24 #include "emb/Types.h"
25
26 #include "lpc/Cntl.h"
27 #include "lpc/Math.h"
28 #include "lpc/Types.h"
29 #include "lpc/Uart.h"
```

```
30
31
32 // local defines
33
34 #define RX_COMMPERIOD (CCLK/10000)
35 #define RX_MATCHSLOT 0
36
37 #define CRC_RST EMB_CLR_CRC32
38
39 #define COMMRANGE 30000
40
41 #define START_STR ">*>"
42 #define BYTECNT_MIN 0
43 #define BYTECNT_MAX 100
44 #define STATE_SENDER_INDEX 0
45 #define RST_PARSE_INDEX 0
46
47 #define HL_CNTL_PKT_ID 'A'
48 #define GLOBAL_PKT_ID 'G'
49 #define LAND_PKT_ID 'L'
50 #define WAY_PKT_ID 'W'
51 #define LL_CNTL_PKT_ID 'D'
52 #define STATE_PKT_ID 'S'
53 #define ENV_PKT_ID 'E'
54 #define HOME_PKT_ID 'H'
55
56 #define LLC_CNTL_PKT_LOOP_CNT 4
57 #define LAND_PKT_LOOP_CNT 4
58 #define HOME_PKT_LOOP_CNT 4
59 #define STATE_PKT_LOOP_CNT 24
```

```

60 #define ENV_PKT_LOOP_CNT 8
61 #define WAY_PKT_LOOP_CNT 20
62 #define GLOBAL_PKT_LOOP_CNT 32
63 #define HL_CNTL_PKT_LOOP_CNT 4
64
65 #define START_STR_PARSE_INDEX 1
66 #define PKT_ID_PARSE_INDEX (START_STR_PARSE_INDEX+1)
67 #define BYTE_CNT_PARSE_INDEX (PKT_ID_PARSE_INDEX+sizeof(
        int32_t))
68 #define HL_CNTL_PKT_PARSE_INDEX 1000
69 #define GLOBAL_PKT_PARSE_INDEX 2000
70 #define LAND_PKT_PARSE_INDEX 3000
71 #define WAY_PKT_PARSE_INDEX 4000
72 #define LL_CNTL_PKT_PARSE_INDEX 5000
73 #define STATE_PKT_PARSE_INDEX 6000
74 #define ENV_PKT_PARSE_INDEX 7000
75 #define HOME_PKT_PARSE_INDEX 8000
76
77
78 // local functions
79
80 void initRxStruct(embComm_s *Comm);
81 void discardCrc(embComm_s *Comm);
82 bool_t isCrcValid(embComm_s *Comm);
83 bool_t isInCommRange(embComm_s *Comm,
84                       embState_s *State);
85 bool_t isUnreadByteInUart(embComm_s *Comm);
86 void parseByteCnt(embComm_s *Comm);
87 void parseEnvPkt(embComm_s *Comm);
88 void parseGlobalPkt(embComm_s *Comm);

```

```

89 void parseHlCntlPkt(embComm_s *Comm);
90 void parseHomePkt(embComm_s *Comm);
91 void parseLandPkt(embComm_s *Comm);
92 void parseLlCntlPkt(embComm_s *Comm);
93 void parsePktId(embComm_s *Comm);
94 void parseStartStr(embComm_s *Comm);
95 void parseStatePkt(embComm_s *Comm);
96 void parseWayPkt(embComm_s *Comm);
97 void syncClocks(embComm_s *Comm,
98                 embState_s *State);
99 void upStateList(embComm_s *Comm,
100                 embState_s *State);
101
102
103 // init rx comm
104
105 void embInitRxComm(embComm_s *Comm)
106 {
107     // build rx structure
108     initRxStruct(Comm);
109
110     // initialize sys match timer
111     lpcInitMatchTmr(&Comm->Rx.Match,
112                   Comm->Tmr);
113 }
114
115
116 // config rx comm
117
118 void embConfigRxComm(embComm_s *Comm)

```

```

119 {
120     // configure rx match
121     lpcSetSlotMatchTmr(&Comm->Rx.Match,
122                       RX_MATCH_SLOT);
123 }
124
125
126 // enable rx comm
127
128 void embEnableRxComm(embComm_s *Comm)
129 {
130     // enable rx match
131     lpcEnableMatchTmr(&Comm->Rx.Match);
132
133     // reset rx match
134     lpcResetMatchTmr(&Comm->Rx.Match,
135                     CCLK);
136 }
137
138
139 // receive comm handle
140
141 void embRxComm(embComm_s *Comm)
142 {
143     // while unread bytes in buf
144     while(isUnreadByteInUart(Comm))
145     {
146         // update parse index and figure out which case
147         switch(++(Comm->Rx.ParseIndex))
148         {

```

```

149         // if start string index
150     case (START_STR_PARSE_INDEX) :
151         parseStartStr (Comm) ;
152         break ;
153
154         // if packet id index
155     case (PKT_ID_PARSE_INDEX) :
156         parsePktId (Comm) ;
157         break ;
158
159         // if byte count index
160     case (BYTE_CNT_PARSE_INDEX) :
161         parseByteCnt (Comm) ;
162         break ;
163
164         // if high level control packet index
165     case (HL_CNTL_PKT_PARSE_INDEX) :
166         parseHlCntlPkt (Comm) ;
167         break ;
168
169         // if position packet index
170     case (GLOBAL_PKT_PARSE_INDEX) :
171         parseGlobalPkt (Comm) ;
172         break ;
173
174         // if land request packet index
175     case (LAND_PKT_PARSE_INDEX) :
176         parseLandPkt (Comm) ;
177         break ;
178

```

```

179         // if waypoint packet index
180     case(WAY_PKT_PARSE_INDEX):
181         parseWayPkt(Comm);
182         break;
183
184         // if low level control packet index
185     case(LL_CNTL_PKT_PARSE_INDEX):
186         parseLlCntlPkt(Comm);
187         break;
188
189         // if state packet index
190     case(STATE_PKT_PARSE_INDEX):
191         parseStatePkt(Comm);
192         break;
193
194         // if environment packet index
195     case(ENV_PKT_PARSE_INDEX):
196         parseEnvPkt(Comm);
197         break;
198
199         // if homing index
200     case(HOME_PKT_PARSE_INDEX):
201         parseHomePkt(Comm);
202         break;
203     }
204 }
205
206 // reset rx comm match
207 lpcResetMatchTmr(&Comm->Rx.Match,
208                 RX_COMMLPERIOD);

```

```

209
210 // clear rx comm match IR
211 lpcClrIrMatchTmr(&Comm->Rx.Match);
212 }
213
214
215 // builds rx comm struct
216
217 void initRxStruct(embComm_s *Comm)
218 { }
219
220
221 // checks if calculated crc is same as read one
222
223 bool_t isCrcValid(embComm_s *Comm)
224 {
225 // local stack
226 int32_t ii;
227 byte_t ReadByte;
228 bool_t DummyBool;
229 int32_t BytesRecv;
230
231 // unread packet for crc
232 lpcOffsetReadUart(&Comm->Uart,
233                  -Comm->Rx.ByteCnt);
234
235 // reset crcs
236 Comm->Rx.CalcCrc = CRC_RST;
237 Comm->Rx.ReadCrc = CRC_RST;
238

```



```

239 // calculate crc of packet
240 for(ii = 0;
241     ii < Comm->Rx.ByteCnt - sizeof(Comm->Rx.CalcCrc);
242     ii++)
243 {
244     lpcReadByteUart(&Comm->Uart ,
245                   &ReadByte ,
246                   &DummyBool);
247     embUpCrc32(&Comm->Rx.CalcCrc ,
248              ReadByte);
249 }
250
251 // read crc from packet
252 LPC_READ_UART(Comm->Uart ,
253              Comm->Rx.ReadCrc ,
254              BytesRecv);
255
256 // return match comparison
257 return((bool_t)(Comm->Rx.CalcCrc == Comm->Rx.ReadCrc));
258 }
259
260
261 // check for valid start string at beginning of packet
262
263 void parseStartStr(embComm_s *Comm)
264 {
265     // local stack
266     int32_t ii;
267     bool_t ValidStartStr;
268     bool_t DummyBool;

```

```

269  byte_t ReadByte;
270
271  // unread bytes for start string comparison
272  lpcOffsetReadUart(&Comm->Uart ,
273                  -(sizeof(STARTSTR)-1));
274
275  // verify start string
276  for(ii = 0, ValidStartStr = true;
277      ii < sizeof(STARTSTR)-1;
278      ii++)
279  {
280      lpcReadByteUart(&Comm->Uart ,
281                    &ReadByte ,
282                    &DummyBool);
283
284      ValidStartStr = ValidStartStr && ((bool_t)(ReadByte ==
285          STARTSTR[ii]));
286
287  // reset parse if not valid start string
288  if(!ValidStartStr)
289  {
290      Comm->Rx.ParseIndex = RST_PARSE_INDEX;
291  }
292 }
293
294
295 // parse byte count from packet
296
297 void parseByteCnt(embComm_s *Comm)

```

```

298 {
299     // local stack
300     int32_t BytesRecv;
301     bool_t ValidByteCnt;
302
303     // read byte count from uart
304     lpcOffsetReadUart(&Comm->Uart,
305                     -sizeof(Comm->Rx.ByteCnt));
306     LPC_READ_UART(Comm->Uart,
307                 Comm->Rx.ByteCnt,
308                 BytesRecv);
309
310     // check if valid byte count
311     ValidByteCnt = (bool_t)((Comm->Rx.ByteCnt >= BYTE_CNT_MIN)
312                            &&
313                            (Comm->Rx.ByteCnt <= BYTE_CNT_MAX))
314
315     ;
316
317     // set parse index if valid byte count
318     if(ValidByteCnt)
319     {
320         // assign corresponding parse index for valid packet id
321         switch(Comm->Rx.PktId)
322         {
323             // if high level control packet
324             case(HL_CNTL_PKT_ID):
325                 Comm->Rx.ParseIndex = HL_CNTL_PKT_PARSE_INDEX;
326                 Comm->Rx.ParseIndex -= Comm->Rx.ByteCnt;
327                 break;

```

```

326         // if global position packet
327     case(GLOBALPKT_ID):
328         Comm->Rx.ParseIndex = GLOBALPKT_PARSEINDEX;
329         Comm->Rx.ParseIndex -= Comm->Rx.ByteCnt;
330         break;
331
332         // if land request packet
333     case(LAND_PKT_ID):
334         Comm->Rx.ParseIndex = LAND_PKT_PARSEINDEX;
335         Comm->Rx.ParseIndex -= Comm->Rx.ByteCnt;
336         break;
337
338         // if waypoint packet
339     case(WAY_PKT_ID):
340         Comm->Rx.ParseIndex = WAY_PKT_PARSEINDEX;
341         Comm->Rx.ParseIndex -= Comm->Rx.ByteCnt;
342         break;
343
344         // if low level control packet
345     case(LL_CNTL_PKT_ID):
346         Comm->Rx.ParseIndex = LL_CNTL_PKT_PARSEINDEX;
347         Comm->Rx.ParseIndex -= Comm->Rx.ByteCnt;
348         break;
349
350         // if state packet
351     case(STATE_PKT_ID):
352         Comm->Rx.ParseIndex = STATE_PKT_PARSEINDEX;
353         Comm->Rx.ParseIndex -= Comm->Rx.ByteCnt;
354         break;
355

```

```

356         // if environment packet
357     case(ENV_PKT_ID):
358         Comm->Rx.ParseIndex = ENV_PKT_PARSE_INDEX;
359         Comm->Rx.ParseIndex -= Comm->Rx.ByteCnt;
360         break;
361
362         // if homing packet
363     case(HOME_PKT_ID):
364         Comm->Rx.ParseIndex = HOME_PKT_PARSE_INDEX;
365         Comm->Rx.ParseIndex -= Comm->Rx.ByteCnt;
366         break;
367
368         // reset if not valid packet
369     default:
370         Comm->Rx.ParseIndex = RST_PARSE_INDEX;
371         break;
372     }
373 }
374
375 // reset parse index if not valid byte count
376 else
377 {
378     Comm->Rx.ParseIndex = RST_PARSE_INDEX;
379 }
380 }
381
382
383 // parse packet id from packet
384
385 void parsePktId(embComm_s *Comm)

```

```

386 {
387     // local stack
388     bool_t DummyBool;;
389
390     // read byte count from uart
391     lpcOffsetReadUart(&Comm->Uart ,
392                     -sizeof(byte_t));
393     lpcReadByteUart(&Comm->Uart ,
394                   &Comm->Rx.PktId ,
395                   &DummyBool);
396 }
397
398
399 // parse low level init packet
400
401 void parseLlCntlPkt(embComm_s *Comm)
402 {
403     // local stack
404     robotId_t TargetRobotId;
405     int32_t BytesRecv;
406     int32_t ii;
407
408     // if valid checksum
409     if(isCrcValid(Comm))
410     {
411         // unread packet for low level control
412         lpcOffsetReadUart(&Comm->Uart ,
413                         -Comm->Rx.ByteCnt);
414
415         // for packet loop entries

```

```

416     for (ii = 0;
417           ii < (Comm->Rx.ByteCnt - sizeof(Comm->Rx.CalcCrc))
418               / LLCNTL_PKT_LOOP_CNT;
419           ii++)
420     {
421         // read robot id
422         LPC_READ_UART(Comm->Uart,
423                       TargetRobotId,
424                       BytesRecv);
425
426         // if me, initialize low level control
427         if (TargetRobotId == Comm->StateList->State[
428             EMB_MY_STATE].RobotId)
429         {
430             Comm->InitLlCntl = true;
431         }
432
433         // discard trailing checksum
434         discardCrc(Comm);
435
436         // reset parse index
437         Comm->Rx.ParseIndex = RST_PARSE_INDEX;
438     }
439
440
441 // parse high level init packet
442
443 void parseHlCntlPkt(embComm_s *Comm)

```

```

444 {
445     // local stack
446     robotId_t TargetRobotId;
447     int32_t BytesRecv;
448     int32_t ii;
449
450     // if valid checksum
451     if(isCrcValid(Comm))
452     {
453         // unread packet for high level control
454         lpcOffsetReadUart(&Comm->Uart,
455                         -Comm->Rx.ByteCnt);
456
457         // for all entries in packet loop
458         for(ii = 0;
459             ii < (Comm->Rx.ByteCnt - sizeof(Comm->Rx.CalcCrc))
460                / HL_CNTL_PKT_LOOP_CNT;
461             ii++)
462         {
463             // read target robot id
464             LPC_READ_UART(Comm->Uart,
465                          TargetRobotId,
466                          BytesRecv);
467
468             // if me, initialize high level control
469             if(TargetRobotId == Comm->StateList->State[
470                EMB_MY_STATE].RobotId)
471             {
472                 Comm->InitHlCntl = true;
473             }
474         }
475     }
476 }

```



```

472     }
473
474     // discard trailing checksum
475     discardCrc(Comm);
476 }
477
478 // reset parse index
479 Comm->Rx.ParseIndex = RST_PARSE_INDEX;
480 }
481
482
483 // parse landing packet
484
485 void parseLandPkt(embComm_s *Comm)
486 {
487     // local stack
488     robotId_t TargetRobotId;
489     int32_t BytesRecv;
490     int32_t ii;
491
492     // if valid checksum
493     if(isCrcValid(Comm))
494     {
495         // unread packet for landing request
496         lpcOffsetReadUart(&Comm->Uart,
497                         -Comm->Rx.ByteCnt);
498
499         // for all entries in packet loop
500         for(ii = 0;

```

```

501         ii < (Comm->Rx.ByteCnt - sizeof(Comm->Rx.CalcCrc))
           / LAND_PKT_LOOP_CNT;
502         ii++)
503     {
504         // read robot id
505         LPC_READ_UART(Comm->Uart,
506                     TargetRobotId,
507                     BytesRecv);
508
509         // if me, initialize landing
510         if(TargetRobotId == Comm->StateList->State[
           EMB_MY_STATE].RobotId)
511         {
512             Comm->InitLand = true;
513         }
514     }
515
516     // discard trailing checksum
517     discardCrc(Comm);
518 }
519
520 // reset parse index
521 Comm->Rx.ParseIndex = RST_PARSE_INDEX;
522 }
523
524
525 // parse homing packet
526
527 void parseHomePkt(embComm_s *Comm)
528 {

```

```

529 // local stack
530 robotId_t TargetRobotId;
531 int32_t BytesRecv;
532 int32_t ii;
533
534 // if valid checksum
535 if(isCrcValid(Comm))
536 {
537     // unread packet for homing request
538     lpcOffsetReadUart(&Comm->Uart,
539                     -Comm->Rx.ByteCnt);
540
541     // for all entires in packet loop
542     for(ii = 0;
543         ii < (Comm->Rx.ByteCnt - sizeof(Comm->Rx.CalcCrc))
544             / HOME_PKT_LOOP_CNT;
545         ii++)
546     {
547         // read robot id
548         LPC_READ_UART(Comm->Uart,
549                     TargetRobotId,
550                     BytesRecv);
551
552         // if me, initialize homing routine
553         if(TargetRobotId == Comm->StateList->State[
554             EMB_MY_STATE].RobotId)
555         {
556             Comm->InitHome = true;
557         }
558     }
559 }

```

```

557
558     // discard trailing checksum
559     discardCrc(Comm);
560 }
561
562 // reset parse index
563 Comm->Rx.ParseIndex = RST_PARSE_INDEX;
564 }
565
566
567 // parse state packet
568
569 void parseStatePkt(embComm_s *Comm)
570 {
571     // local stack
572     embState_s State;
573     bool_t CommRange;
574     int32_t ii;
575     int32_t jj;
576     int32_t BytesRecv;
577
578     // if valid checksum
579     if(isCrcValid(Comm))
580     {
581         // unread packet for state information
582         lpcOffsetReadUart(&Comm->Uart,
583             -Comm->Rx.ByteCnt);
584
585         // for all entires in packet loop
586         for(ii = 0, CommRange = false;

```

```

587         ii < (Comm->Rx.ByteCnt - sizeof(Comm->Rx.CalcCrc))
           / STATE_PKT_LOOP_CNT;
588     ii++)
589     {
590         // read robot id
591         LPC_READ_UART(Comm->Uart,
592                     State.RobotId,
593                     BytesRecv);
594
595         // read time stamp
596         LPC_READ_UART(Comm->Uart,
597                     State.TimeStamp,
598                     BytesRecv);
599
600         // read state positions
601         for(jj = 0;
602            jj < sizeof(State.Pos)/sizeof(State.Pos[0]);
603            jj++)
604         {
605             LPC_READ_UART(Comm->Uart,
606                         State.Pos[jj],
607                         BytesRecv);
608         }
609
610         // if info of sender, will only work for index==0
611         if(ii == STATE_SENDER_INDEX)
612         {
613             // see if sender was in comm range
614             CommRange = isInCommRange(Comm,
615                                       &State);

```

```

616
617         // if in comm range, sync clocks
618         if (CommRange)
619             {
620                 /*
621                 embSyncClock (Comm->Clock ,
622                               State.TimeStamp + Comm->
623                               Latency ,
624                               Comm->Rx.Lambda);
625                 */
626             }
627
628         // if in comm range, update state list
629         if (CommRange)
630             {
631                 upStateList (Comm,
632                              &State);
633             }
634     }
635
636     // discard trailing checksum
637     discardCrc (Comm);
638 }
639
640 // reset parse index
641 Comm->Rx.ParseIndex = RST_PARSE_INDEX;
642 }
643
644

```

```

645 // parse global position packet
646
647 void parseGlobalPkt(embComm_s *Comm)
648 {
649     // local stack
650     robotId_t TargetRobotId;
651     embGlobal_s Global;
652     int32_t ii;
653     int32_t jj;
654     int32_t BytesRecv;
655
656     // if valid checksum
657     if(isCrcValid(Comm))
658     {
659         Comm->Status->Yellow.Speed = LED_Slow;
660
661         // unread packet for state information
662         lpcOffsetReadUart(&Comm->Uart,
663             -Comm->Rx.ByteCnt);
664
665         // for all entires in packet loop
666         for(ii = 0;
667             ii < (Comm->Rx.ByteCnt - sizeof(Comm->Rx.CalcCrc))
668                 / GLOBALPKTLOOP_CNT;
669             ii++)
670         {
671             // read robot id
672             LPC_READ_UART(Comm->Uart,
673                 TargetRobotId,
674                 BytesRecv);

```

```

674
675     // read time stamp
676     LPC_READ_UART(Comm->Uart ,
677                 Global.TimeStamp ,
678                 BytesRecv);
679
680     // read state positions
681     for(jj = 0;
682         jj < sizeof(Global.So3)/sizeof(Global.So3[0]);
683         jj++)
684     {
685         LPC_READ_UART(Comm->Uart ,
686                     Global.So3[jj] ,
687                     BytesRecv);
688     }
689
690     // if my global state , keep it
691     if(TargetRobotId == Comm->StateList->State[
        EMB_MY_STATE].RobotId)
692     {
693         Comm->Status->Yellow.Speed = LED_Flkr;
694
695         // update global
696         embUpGlobal(Comm->Global ,
697                   &Global);
698
699         // update state
700         embUpStateFromGlobal(Comm->Global);
701
702         // update pid

```



```

703             embUpPidFromGlobal(Comm->Global);
704         }
705     }
706
707     // discard trailing checksum
708     discardCrc(Comm);
709 }
710
711 // reset parse index
712 Comm->Rx.ParseIndex = RST_PARSE_INDEX;
713 }
714
715
716 // parse environment initialization packet
717
718 void parseEnvPkt(embComm_s *Comm)
719 {
720     // local stack
721     embEnv_s Env;
722     int32_t ii;
723     int32_t jj;
724     int32_t BytesRecv;
725
726     // if valid checksum
727     if(isCrcValid(Comm))
728     {
729         // unread packet for environment information
730         lpcOffsetReadUart(&Comm->Uart,
731             -Comm->Rx.ByteCnt);
732

```

```

733     // clear environment information
734     embClrEnv(Comm->Env);
735
736     // for all entires in packet loop
737     for(ii = 0;
738         (ii < (Comm->Rx.ByteCnt - sizeof(Comm->Rx.CalcCrc))
739          / ENV_PKT_LOOP_CNT) && (ii <
740          EMB_MAX_BNDRY_SIZE_ENV);
739         ii++)
740     {
741         // read environment entry
742         for(jj = 0;
743             jj < sizeof(Env.Bndry[0].Pos)/sizeof(Env.Bndry
744                 [0].Pos[0]);
745             jj++)
746         {
747             LPC_READ_UART(Comm->Uart,
748                           Env.Bndry[ii].Pos[jj],
749                           BytesRecv);
750         }
751         // initialize environment
752         Comm->InitEnv = true;
753     }
754
755     // discard trailing checksum
756     discardCrc(Comm);
757 }
758
759 // reset parse index

```

```

760     Comm->Rx.ParseIndex = RST_PARSE_INDEX;
761 }
762
763
764 // parse waypoint packet
765
766 void parseWayPkt(embComm_s *Comm)
767 {
768     // local stack
769     robotId_t TargetRobotId;
770     int32_t BytesRecv;
771     pos_t Way[4];
772     int32_t ii;
773     int32_t jj;
774
775     // if valid checksum
776     if(isCrcValid(Comm))
777     {
778         // unread packet for landing request
779         lpcOffsetReadUart(&Comm->Uart,
780                         -Comm->Rx.ByteCnt);
781
782         // for all entries in packet loop
783         for(ii = 0;
784             ii < (Comm->Rx.ByteCnt - sizeof(Comm->Rx.CalcCrc))
785                / WAY_PKT_LOOP_CNT;
786             ii++)
787         {
788             // read robot id
789             LPC_READ_UART(Comm->Uart,

```

```

789             TargetRobotId ,
790             BytesRecv);
791
792     // read environment entry
793     for(jj = 0;
794         jj < sizeof(Way)/sizeof(Way[0]);
795         jj++)
796     {
797         LPC_READ_UART(Comm->Uart ,
798                     Way[jj] ,
799                     BytesRecv);
800     }
801
802     // if for me, write to waypoint struct
803     if(TargetRobotId == Comm->StateList->State[
804         EMB_MY_STATE].RobotId)
805     {
806         embUpWayGlobal(Comm->Global ,
807                     Way);
808     }
809
810     // discard trailing checksum
811     discardCrc(Comm);
812 }
813
814 // reset parse index
815 Comm->Rx.ParseIndex = RST_PARSE_INDEX;
816 }
817

```

```

818
819 // discard trailing checksum
820
821 void discardCrc(embComm_s *Comm)
822 {
823     // local stack
824     crc32_t Crc;
825     int32_t BytesRecv;
826
827     // discard
828     LPC_READ_UART(Comm->Uart,
829                  Crc,
830                  BytesRecv);
831 }
832
833
834 // check if in comm range
835
836 bool_t isInCommRange(embComm_s *Comm,
837                      embState_s *State)
838 {
839     // local stack
840     int32_t Dist;
841
842     // calc distance to sender
843     Dist = int32Dist(Comm->StateList->State[EMB_MY.STATE].Pos,
844                   State->Pos,
845                   3);
846
847     // if in comm range. return true, else false

```

```

848  if( Dist < COMMRANGE)
849      {
850          return(true);
851      }
852  else
853      {
854          return( false);
855      }
856  }
857
858
859  // updates state list
860
861  void upStateList(embComm_s *Comm,
862                  embState_s *State)
863  {
864      // local stack
865      int32_t Index;
866      int32_t ii;
867      bool_t IsValid;
868      int32_t TimeStampDiff;
869
870      // find robot index
871      IsValid = embFindState(Comm->StateList ,
872                            State->RobotId ,
873                            &Index);
874
875      // if valid and not my state
876      if((IsValid) && (Index != EMB_MY_STATE))
877      {

```

```

878     // if more current, update list
879     TimeStampDiff = State->TimeStamp - Comm->StateList->
        State[Index].TimeStamp;
880
881     if(TimeStampDiff > 0)
882     {
883         Comm->StateList->State[Index].TimeStamp = State->
            TimeStamp;
884         for(ii = 0;
885             ii < sizeof(State->Pos)/sizeof(State->Pos[0]);
886             ii++)
887         {
888             Comm->StateList->State[Index].Pos[ii] = State->
                Pos[ii];
889         }
890     }
891 }
892 }
893
894
895 // see if unread bytes still in uart
896
897 bool_t isUnreadByteInUart(embComm_s *Comm)
898 {
899     // local stack
900     bool_t ByteRecv;
901     byte_t DummyByte;
902
903     // function
904     lpcReadByteUart(&Comm->Uart,

```

```
905             &DummyByte,  
906             &ByteRecv);  
907  
908     // return bool  
909     return(ByteRecv);  
910 }
```



```
1 /*
2  * emb/TxAp.c
3  *
4  * tx ap source code for drl quad-rotor project
5  *
6  * Brian J. Julian
7  *
8  * bjulian{at}mit{dot}edu
9  *
10 * Version 0.1
11 *
12 * 31 March 2009
13 *
14 */
15
16
17 // includes
18
19 #include "emb/Clock.h"
20 #include "emb/Ap.h"
21 #include "emb/Crc16.h"
22 #include "emb/Nav.h"
23 #include "lpc/Buf.h"
24 #include "lpc/Cntl.h"
25 #include "lpc/Math.h"
26
27
28 // local defines
29
30 #define CRC_RST EMB_CLR_CRC32
```

```

31
32 #define TX_MATCHSLOT 3
33 #define TX_AP_PERIOD (CCLK)
34
35 #define START_STR ">*>"
36 #define STATE_PKT_ID 'S'
37 #define STATE_PKT_LOOP_CNT 10
38
39 #define LAT_ORIGIN 423599810
40 #define LONG_ORIGIN -710891290
41
42 #define HEIGHT_ORIGIN 3000
43
44 #define LAT_LENGTH 11
45 #define LONG_LENGTH 8
46
47
48 // local functions
49
50 void calcApBufCrc(embAp_s *Ap);
51 void initTxStructAp(embTxAp_s *Tx);
52 void sendState(embAp_s *Ap);
53 void sendWay(embAp_s *Ap);
54 void sendCmd(embAp_s *Ap);
55
56 // initialize tx ap
57
58 void embInitTxAp(embAp_s *Ap)
59 {
60 // initialize tx ap struct

```

```

61  initTxStructAp(&Ap->Tx) :
62
63  // initialize tx ap buf
64  lpcInitBuf(&Ap->Tx.Buf) ;
65
66  // initialize sys match timer
67  lpcInitMatchTmr(&Ap->Tx.Match ,
68                  Ap->Tmr) ;
69 }
70
71
72 // configure tx ap
73
74 void embConfigTxAp(embAp_s *Ap)
75 {
76  // assign read buf
77  lpcAssignBuf(&Ap->Tx.Buf ,
78              Ap->Tx.BufMem ,
79              LPC_TX_BUF_SIZE_UART) ;
80
81  // configure tx match
82  lpcSetSlotMatchTmr(&Ap->Tx.Match ,
83                    TX_MATCH_SLOT) ;
84 }
85
86
87 // enable tx ap
88
89 void embEnableTxAp(embAp_s *Ap)
90 {

```

```

91  // enable read buf
92  lpcEnableBuf(&Ap->Tx.Buf);
93
94  // enable tx match tmr
95  lpcEnableMatchTmr(&Ap->Tx.Match);
96
97  // reset tx ap match
98  lpcResetMatchTmr(&Ap->Tx.Match,
99                  CCLK);
100 }
101
102
103 // main tx ap handle
104
105 void embTxAp(embAp.s *Ap)
106 {
107     // local stack
108
109     // action based on navigation setting
110     switch(Ap->Nav->Flight)
111     {
112         // if indoor flight, send commands
113         case(NAV_Indoor):
114             sendCmd(Ap);
115             break;
116
117         // if outdoor flight, send ways
118         case(NAV_Outdoor):
119             sendWay(Ap);
120             break;

```

```

121     }
122
123     // reset tx ap match
124     lpcResetMatchTmr(&Ap->Tx.Match,
125                     TX_AP_PERIOD);
126
127     // clear tx ap match IR
128     lpcClrIrMatchTmr(&Ap->Tx.Match);
129 }
130
131
132 // calculate crc of buffer
133
134 void calcApBufCrc(embAp_s *Ap)
135 {
136     // local stack
137     byte_t Read;
138     bool_t ByteRecv;
139     int32_t OffsetSize;
140
141     // offset non crc portion of packet
142     OffsetSize = sizeof(START_STR) - 1;
143     OffsetSize += sizeof(byte_t);
144     OffsetSize += sizeof(Ap->Tx.ByteCnt);
145
146     lpcOffsetReadBuf(&Ap->Tx.Buf,
147                    OffsetSize);
148
149     // reset Crc
150     Ap->Tx.Crc = EMB_CLR_CRC16;

```

```

151
152  // calc Crc of buf
153  lpcReadByteBuf(&Ap->Tx.Buf,
154                &Read,
155                &ByteRecv);
156
157  while( ByteRecv)
158  {
159      embUpCrc16(&Ap->Tx.Crc,
160               Read);
161      lpcReadByteBuf(&Ap->Tx.Buf,
162                   &Read,
163                   &ByteRecv);
164      OffsetSize++;
165  }
166
167  // unread buf
168  lpcOffsetReadBuf(&Ap->Tx.Buf,
169                 -OffsetSize);
170 }
171
172
173 // initialize tx structure
174
175 void initTxStructAp(embTxAp_s *Tx)
176 { }
177
178
179 // send way over ap uart
180

```

```

181 void sendWay(embAp_s *Ap)
182 {
183     // local stack
184     uint8_t  WpNumber = 1;
185     uint8_t  Dummy1 = 0;
186     uint16_t Dummy2 = 0;
187     uint8_t  Properties = ((0x01)|(0x02)|(0x04)|(0x10));
188     uint8_t  MaxSpeed = 20;
189     uint16_t Time = 0;
190     uint16_t PosAcc = 2500;
191     int16_t  ChkSum;
192     int32_t  X;
193     int32_t  Y;
194     int32_t  Yaw = 0;
195     int32_t  Height = 0;
196
197     // Position calculation
198     X = (Ap->Global->Way[0]/LONGLENGTH)+LONG_ORIGIN;
199     Y = (Ap->Global->Way[1]/LATLENGTH)+LAT_ORIGIN;
200     Yaw = -(180000*Ap->Global->Way[3])/INT_PI;
201     Height = Ap->Global->Way[2];
202
203
204     // Checksum calcualtion
205     ChkSum = (0xAAAA) + (int16_t)Yaw + (int16_t)Height + (
                int16_t)Time +
206     (int16_t)X + (int16_t)Y + (int16_t)MaxSpeed + (int16_t)
                PosAcc +
207     (int16_t)Properties + (int16_t)WpNumber;
208

```

```

209
210 // write way cmd to buffer
211 LPC_WRITE_BUF(Ap->Tx.Buf, ">*>ws");
212 LPC_WRITE_BUF(Ap->Tx.Buf, WpNumber);
213 LPC_WRITE_BUF(Ap->Tx.Buf, Dummy1);
214 LPC_WRITE_BUF(Ap->Tx.Buf, Dummy2);
215 LPC_WRITE_BUF(Ap->Tx.Buf, Properties);
216 LPC_WRITE_BUF(Ap->Tx.Buf, MaxSpeed);
217 LPC_WRITE_BUF(Ap->Tx.Buf, Time);
218 LPC_WRITE_BUF(Ap->Tx.Buf, PosAcc);
219 LPC_WRITE_BUF(Ap->Tx.Buf, ChkSum);
220 LPC_WRITE_BUF(Ap->Tx.Buf, X);
221 LPC_WRITE_BUF(Ap->Tx.Buf, Y);
222 LPC_WRITE_BUF(Ap->Tx.Buf, Yaw);
223 LPC_WRITE_BUF(Ap->Tx.Buf, Height);
224
225 // write buf to uart
226 lpcWriteBufToUart(&Ap->Uart,
227                  &Ap->Tx.Buf);
228 }
229
230 // send way over ap uart
231
232 void sendCmd(embAp_s *Ap)
233 {
234 // local stack
235 int16_t ChkSum;
236 int16_t Property = 0x02;
237 uint16_t TempShort = 0x0F;
238

```



```

239 // Checksum calculation
240 ChkSum = (0xAAAA);
241 ChkSum += (int16_t)Ap->PidList->Input [0];
242 ChkSum += (int16_t)Ap->PidList->Input [1];
243 ChkSum += (int16_t)Ap->PidList->Input [2];
244 ChkSum += (int16_t)Ap->PidList->Input [3];
245 ChkSum += (int16_t)TempShort;
246
247 // write way cmd to buffer
248 LPC_WRITE_BUF(Ap->Tx. Buf, ">*>p");
249 LPC_WRITE_BUF(Ap->Tx. Buf, Property);
250 LPC_WRITE_BUF(Ap->Tx. Buf, ">*>di");
251 LPC_WRITE_BUF(Ap->Tx. Buf, Ap->PidList->Input [0]);
252 LPC_WRITE_BUF(Ap->Tx. Buf, Ap->PidList->Input [1]);
253 LPC_WRITE_BUF(Ap->Tx. Buf, Ap->PidList->Input [2]);
254 LPC_WRITE_BUF(Ap->Tx. Buf, Ap->PidList->Input [3]);
255 LPC_WRITE_BUF(Ap->Tx. Buf, TempShort);
256 LPC_WRITE_BUF(Ap->Tx. Buf, ChkSum);
257
258
259 // write buf to uart
260 lpcWriteBufToUart(&Ap->Uart,
261                  &Ap->Tx. Buf);
262 }

```

```
1 /*
2  * emb/TxComm.c
3  *
4  * tx communication source code for drl quad-rotor project
5  *
6  * Brian J. Julian
7  *
8  * bjulian{at}mit{dot}edu
9  *
10 * Version 0.1
11 *
12 * 31 March 2009
13 *
14 */
15
16
17 // includes
18
19 #include "emb/Clock.h"
20 #include "emb/Comm.h"
21 #include "emb/Crc32.h"
22 #include "emb/Nav.h"
23 #include "lpc/Buf.h"
24 #include "lpc/Cntl.h"
25
26
27 // local defines
28
29 #define CRC_RST EMB_CLR_CRC32
30
```

```

31 #define TX_MATCHSLOT 1
32 #define TX_COMMPERIOD (CCLK)
33
34 #define START_STR ">*>"
35 #define STATE_PKT_ID 'S'
36 #define STATE_PKT_LOOP_CNT 10
37
38 #define FAKE_STR "abc_"
39
40
41 // local functions
42
43 void calcBufCrc(embComm_s *Comm);
44 void initTxStruct(embTxComm_s *Tx);
45 void sendState(embComm_s *Comm);
46
47
48 // initialize tx comm
49
50 void embInitTxComm(embComm_s *Comm)
51 {
52     // initialize tx comm struct
53     initTxStruct(&Comm->Tx);
54
55     // initialize tx comm buf
56     lpcInitBuf(&Comm->Tx.Buf);
57
58     // initialize sys match timer
59     lpcInitMatchTmr(&Comm->Tx.Match,
60                    Comm->Tmr);

```

```

61 }
62
63
64 // configure tx comm
65
66 void embConfigTxComm(embComm_s *Comm)
67 {
68     // assign read buf
69     lpcAssignBuf(&Comm->Tx.Buf,
70                 Comm->Tx.BufMem,
71                 LPC_TX_BUF_SIZE_UART);
72
73     // configure tx match
74     lpcSetSlotMatchTmr(&Comm->Tx.Match,
75                        TX_MATCH_SLOT);
76 }
77
78
79 // enable tx comm
80
81 void embEnableTxComm(embComm_s *Comm)
82 {
83     // enable read buf
84     lpcEnableBuf(&Comm->Tx.Buf);
85
86     // enable tx match tmr
87     lpcEnableMatchTmr(&Comm->Tx.Match);
88
89     // reset tx comm match
90     lpcResetMatchTmr(&Comm->Tx.Match,

```

```

91             CCLK);
92 }
93
94
95 // main tx comm handle
96
97 void embTxComm(embComm_s *Comm)
98 {
99     // local stack
100
101     // action based on navigation setting
102     switch(Comm->Nav->Flight)
103     {
104         // if indoor flight, do nothing
105         case(NAV_Indoor):
106             break;
107
108         // if outdoor flight, send states
109         case(NAV_Outdoor):
110             sendState(Comm);
111             break;
112     }
113
114     // reset tx comm match
115     lpcResetMatchTmr(&Comm->Tx.Match,
116                     TX.COMMPERIOD);
117
118     // clear tx comm match IR
119     lpcClrIrMatchTmr(&Comm->Tx.Match);
120 }

```

```

121
122
123 // calculate crc of buffer
124
125 void calcBufCrc(embComm_s *Comm)
126 {
127     // local stack
128     byte_t Read;
129     bool_t ByteRecv;
130     int32_t OffsetSize;
131
132     // offset non crc portion of packet
133     OffsetSize = sizeof(START_STR) - 1;
134     OffsetSize += sizeof(byte_t);
135     OffsetSize += sizeof(Comm->Tx.ByteCnt);
136
137     lpcOffsetReadBuf(&Comm->Tx.Buf,
138                    OffsetSize);
139
140     // reset Crc
141     Comm->Tx.Crc = EMB_CLR_CRC32;
142
143     // calc Crc of buf
144     lpcReadByteBuf(&Comm->Tx.Buf,
145                  &Read,
146                  &ByteRecv);
147
148     while(ByteRecv)
149     {
150         embUpCrc32(&Comm->Tx.Crc,

```

```

151             Read);
152     lpcReadByteBuf(&Comm->Tx.Buf,
153                 &Read,
154                 &ByteRecv);
155     OffsetSize++;
156 }
157
158 // unread buf
159 lpcOffsetReadBuf(&Comm->Tx.Buf,
160                -OffsetSize);
161 }
162
163
164 // initialize tx structure
165
166 void initTxStruct(embTxComm_s *Tx)
167 { }
168
169
170 // send state over comm uart
171
172 void sendState(embComm_s *Comm)
173 {
174     // local stack
175     int32_t ii;
176     int32_t jj;
177
178     // calc byte count
179     Comm->Tx.ByteCnt = sizeof(Comm->StateList->State[0].RobotId
    );

```

```

180  Comm->Tx.ByteCnt += sizeof(Comm->StateList->State[0].
        TimeStamp);
181  Comm->Tx.ByteCnt += sizeof(Comm->StateList->State[0].Pos);
182  Comm->Tx.ByteCnt *= Comm->Mhop->SlotLength;
183  Comm->Tx.ByteCnt += sizeof(Comm->Tx.Crc);
184
185  // write start string
186  lpcWriteBuf(&Comm->Tx.Buf,
187             (byte_t *)START_STR,
188             sizeof(START_STR)-1);
189
190  // write state packet id
191  lpcWriteByteBuf(&Comm->Tx.Buf,
192                (byte_t)STATE_PKT_ID);
193
194  // write byte count
195  LPC_WRITE_BUF(Comm->Tx.Buf,
196                Comm->Tx.ByteCnt);
197
198  // get my time stamp
199  Comm->StateList->State[0].TimeStamp = embGetClock(Comm->
        Clock);
200
201  // write states
202  for(ii = 0;
203      ii < Comm->Mhop->SlotLength;
204      ii++)
205      {
206      LPC_WRITE_BUF(Comm->Tx.Buf,

```



```

207             Comm->StateList->State [Comm->StateList->
                SendIndex [ ii ]]. RobotId);
208     LPC_WRITE_BUF(Comm->Tx. Buf,
209             Comm->StateList->State [Comm->StateList->
                SendIndex [ ii ]]. TimeStamp);
210     for (jj = 0;
211         jj < sizeof(Comm->StateList->State [0]. Pos) / sizeof
                (Comm->StateList->State [0]. Pos [0]);
212         jj++)
213     {
214         LPC_WRITE_BUF(Comm->Tx. Buf,
215             Comm->StateList->State [Comm->
                StateList->SendIndex [ ii ]]. Pos [ jj ]
                ;
216     }
217 }
218
219 // calc crc
220 calcBufCrc (Comm);
221
222 // write crc to buf
223 LPC_WRITE_BUF(Comm->Tx. Buf,
224             Comm->Tx. Crc);
225
226 // write buf to uart
227 lpcWriteBufToUart(&Comm->Uart,
228             &Comm->Tx. Buf);
229 }

```

```
1 /*
2  * emb/State.h
3  *
4  * state includes for the drl quad-rotor project
5  *
6  * Brian J. Julian
7  *
8  * bjulian{at}mit{dot}edu
9  *
10 * Version 0.1
11 *
12 * 31 March 2009
13 *
14 */
15
16
17 // __EMB_STATE_H__
18
19 #ifndef __EMB_STATE_H__
20 #define __EMB_STATE_H__
21
22
23 // includes
24
25 #include "emb/Types.h"
26 #include "lpc/Types.h"
27
28
29 // defines
30
```

```

31 #define EMB_MY_STATE 0
32 #define EMB_MAX_SIZE_STATE 10
33
34 #define EMB_X_STATE 0
35 #define EMB_Y_STATE 1
36 #define EMB_Z_STATE 2
37 #define EMB_YAW_STATE 3
38
39 // state struct
40
41 typedef struct
42 {
43     clock_t TimeStamp;
44     robotId_t RobotId;
45     pos_t Pos[4];
46 }
47     embState_s;
48
49
50 // state list struct
51
52 typedef struct
53 {
54     int32_t Size;
55     embState_s State[EMB_MAX_SIZE_STATE];
56     int32_t SendIndex[EMB_MAX_SIZE_STATE];
57 }
58     embStateList_s;
59
60

```

```
61 // functions
62
63 void embInitState(embStateList_s *StateList);
64 void embConfigState(embStateList_s *StateList);
65 void embEnableState(embStateList_s *StateList);
66
67 bool_t embFindState(const embStateList_s *StateList,
68                    const robotId_t RobotId,
69                    int32_t *Index);
70 robotId_t embRobotIdState(robotName_t RobotName);
71
72
73 // __EMB_STATE_H__
74
75 #endif
```

```
1 /*
2  * emb/State.c
3  *
4  * state source code for the drl quad-rotor project
5  *
6  * Brian J. Julian
7  *
8  * bjulian{at}mit{dot}edu
9  *
10 * Version 0.1
11 *
12 * 31 March 2009
13 *
14 */
15
16
17 // includes
18
19 #include "emb/State.h"
20 #include "emb/Types.h"
21
22 #include "lpc/Math.h"
23 #include "lpc/Types.h"
24
25
26 // local defines
27
28 #define MAX_ROBOT_NAME 7
29 #define ROBOT_NAME_BASE 27
30
```

```

31
32 // initialize state
33
34 void embInitState(embStateList_s *StateList)
35 { }
36
37
38 // configure state
39
40 void embConfigState(embStateList_s *StateList)
41 { }
42
43
44 // enable state
45
46 void embEnableState(embStateList_s *StateList)
47 { }
48
49
50 // returns robot id from robot name
51
52 robotId_t embRobotIdState(robotName_t RobotName)
53 {
54     // local stack
55     robotId_t RobotId;
56     int ii;
57
58     // zero robot id
59     RobotId = 0;
60

```

```

61 // calc robot id
62 for(ii = 0;
63     (RobotName[ii] != '\0') && (ii < MAXROBOTNAME);
64     ii++)
65     {
66     RobotId += ((robotId_t)(RobotName[ii]-'a'+1))*int32Pow(
        ROBOT_NAME_BASE, ii);
67     }
68
69 // return id
70 return(RobotId);
71 }
72
73
74 // finds robot name among state list
75
76 bool_t embFindState(const embStateList_s *StateList,
77                    const robotId_t RobotId,
78                    int32_t *Index)
79 {
80 // find in state list
81 for(*Index = 0;
82     (StateList->State[*Index].RobotId != RobotId) && ((*
        Index) < StateList->Size);
83     (*Index)++);
84
85 if((*Index) == EMB_MAX_SIZE_STATE)
86     {
87     return(false);
88     }

```

```
89     else
90     {
91         return(true);
92     }
93 }
```



```
1 /*
2  * emb/Status.h
3  *
4  * status includes for the drl quad-rotor project
5  *
6  * Brian J. Julian
7  *
8  * bjulian{at}mit{dot}edu
9  *
10 * Version 0.1
11 *
12 * 31 March 2009
13 *
14 */
15
16
17 // __EMB_STATUS_H__
18
19 #ifndef __EMB_STATUS_H__
20 #define __EMB_STATUS_H__
21
22
23 // includes
24
25 #include "lpc/Types.h"
26 #include "lpc/Tmr.h"
27
28
29 // led speed enum
30
```

```

31 typedef enum
32     {
33         LED_On,
34         LED_Slow,
35         LED_Med,
36         LED_Fast,
37         LED_Flkr
38     }
39     embLedSpeed_e;
40
41
42 // led struct
43
44 typedef struct
45     {
46         lpcMatchTmr_s Match;
47         embLedSpeed_e Speed;
48     }
49     embLed_s;
50
51
52 // status struct
53
54 typedef struct
55     {
56         lpcTmr_s Tmr;
57
58         embLed_s Green;
59         embLed_s Yellow;
60         embLed_s Red;

```

```
61 }
62     embStatus_s;
63
64
65 // functions
66
67 void embInitStatus(embStatus_s *Status);
68 void embConfigStatus(embStatus_s *Status);
69 void embEnableStatus(embStatus_s *Status);
70
71
72 // __EMB_STATUS_H__
73
74 #endif
```

```
1 /*
2  * emb/Status.c
3  *
4  * status source code for the drl quad-rotor project
5  *
6  * Brian J. Julian
7  *
8  * bjulian{at}mit{dot}edu
9  *
10 * Version 0.1
11 *
12 * 31 March 2009
13 *
14 */
15
16
17 // includes
18
19 #include "emb/Status.h"
20 #include "lpc/Cntl.h"
21 #include "lpc/Tmr.h"
22 #include "lpc/Types.h"
23
24
25 // local defines
26
27 #define STATUS_TMR_NUM 1
28
29 #define GREEN_MATCH_SLOT 3
30 #define GREEN_MATCH_PIN 20
```

```

31
32 #define YELLOW_MATCHSLOT 2
33 #define YELLOW_MATCHPIN 19
34
35 #define RED_MATCHSLOT 1
36 #define RED_MATCHPIN 13
37
38 #define ON_PERIOD (CCLK/100)
39 #define OFF_PERIOD CCLK
40 #define SLOW_PERIOD CCLK
41 #define MED_PERIOD (CCLK/4)
42 #define FAST_PERIOD (CCLK/16)
43
44
45 // global pointer
46
47 embStatus_s *LinkedStatus;
48
49
50 // local functions
51
52 void manageLedState(embLed_s *Led,
53                    lpcTmr_s *Tmr);
54 void statusTmrVectAddr(void) __attribute__((interrupt("IRQ")))
55    );
56
57 // initialize status
58
59 void embInitStatus(embStatus_s *Status)

```

```

60 {
61     // link status struct
62     LinkedStatus = Status;
63
64     // initialize status timer
65     lpcInitTmr(&Status->Tmr);
66
67     // initialize green led
68     Status->Green.Speed = LED_On;
69     lpcInitMatchTmr(&Status->Green.Match,
70                    &Status->Tmr);
71
72     // initialize yellow led
73     Status->Yellow.Speed = LED_On;
74     lpcInitMatchTmr(&Status->Yellow.Match,
75                    &Status->Tmr);
76
77     /*
78     // initialize red led
79     Status->Red.Speed = LED_On;
80     lpcInitMatchTmr(&Status->Red.Match,
81                    &Status->Tmr);
82     */
83 }
84
85
86 // configure status
87
88 void embConfigStatus(embStatus_s *Status)
89 {

```

```

90 // configure status timer
91 lpcSetNumTmr(&Status->Tmr,
92             STATUS_TMR_NUM);
93 lpcSetVectAddrTmr(&Status->Tmr,
94                  (reg32_t)statusTmrVectAddr);
95
96 // configure green led
97 lpcSetSlotMatchTmr(&Status->Green.Match,
98                  GREEN_MATCH_SLOT);
99 lpcSetPinMatchTmr(&Status->Green.Match,
100                 GREEN_MATCH_PIN);
101
102 // configure yellow led
103 lpcSetSlotMatchTmr(&Status->Yellow.Match,
104                  YELLOW_MATCH_SLOT);
105 lpcSetPinMatchTmr(&Status->Yellow.Match,
106                 YELLOW_MATCH_PIN);
107
108 /*
109 // configure red led
110 lpcSetSlotMatchTmr(&Status->Red.Match,
111                  RED_MATCH_SLOT);
112 lpcSetPinMatchTmr(&Status->Red.Match,
113                 RED_MATCH_PIN);
114 */
115 }
116
117
118 // enable status
119

```

```

120 void embEnableStatus(embStatus_s *Status)
121 {
122     // enable status timer
123     lpcEnableTmr(&Status->Tmr);
124
125     // enable green led
126     lpcSetExtMatchTmr(&Status->Green.Match);
127     lpcEnableMatchTmr(&Status->Green.Match);
128     lpcResetMatchTmr(&Status->Green.Match,
129                     CCLK);
130
131     // enable Yellow led
132     lpcSetExtMatchTmr(&Status->Yellow.Match);
133     lpcEnableMatchTmr(&Status->Yellow.Match);
134     lpcResetMatchTmr(&Status->Yellow.Match,
135                     CCLK);
136
137     /*
138     // enable Red led
139     lpcSetExtMatchTmr(&Status->Red.Match);
140     lpcEnableMatchTmr(&Status->Red.Match);
141     lpcResetMatchTmr(&Status->Red.Match,
142                     CCLK);
143     */
144 }
145
146
147 // timer management of leds
148
149 void manageLedState(embLed_s *Led,

```



```

150             lpcTmr_s *Tmr)
151 {
152     // local stack
153     bool_t Read;
154     reg32_t TcOffset;
155
156     // adjust match tc depending on set speed
157     switch(Led->Speed)
158     {
159         // case on
160         case(LED_On):
161             TcOffset = ON_PERIOD;
162             break;
163
164         // case flkr
165         case(LED_Flkr):
166             lpcReadPinMatchTmr(&Led->Match,
167                                 &Read);
168             if(Read)
169             {
170                 TcOffset = ON_PERIOD;
171             }
172             else
173             {
174                 TcOffset = OFF_PERIOD;
175             }
176             break;
177
178         // case slow
179         case(LED_Slow):

```

```

180     TcOffset = SLOW_PERIOD;
181     break;
182
183     // case medium
184     case(LED_Med):
185         TcOffset = MED_PERIOD;
186         break;
187
188     // case fast
189     case(LED_Fast):
190         TcOffset = FAST_PERIOD;
191         break;
192
193     // default
194     default:
195         TcOffset = CCLK;
196         break;
197     }
198
199     // reset match
200     lpcResetMatchTmr(&Led->Match,
201                    TcOffset);
202 }
203
204
205 // timer vector address handle
206
207 void statusTmrVectAddr(void)
208 {
209     // local stack

```

```

210  embStatus_s *Status;
211  bool_t Read;
212
213  // create status link
214  Status = LinkedStatus;
215
216  // if green match caused IR
217  lpcReadIrMatchTmr(&Status->Green.Match,
218                  &Read);
219  if(Read)
220  {
221      manageLedState(&Status->Green,
222                  &Status->Tmr);
223      lpcClrIrMatchTmr(&Status->Green.Match);
224  }
225
226  // if yellow match caused IR
227  lpcReadIrMatchTmr(&Status->Yellow.Match,
228                  &Read);
229  if(Read)
230  {
231      manageLedState(&Status->Yellow,
232                  &Status->Tmr);
233      lpcClrIrMatchTmr(&Status->Yellow.Match);
234  }
235
236  /*
237  // if red match caused IR
238  lpcReadIrMatchTmr(&Status->Red.Match,
239                  &Read);

```

```
240
241  if (Read)
242    {
243      manageLedState(&Status->Red,
244                &Status->Tmr);
245      lpcClrIrMatchTmr(&Status->Red.Match);
246    }
247  */
248
249  // reset vic
250  lpcResetVicTmr(&Status->Tmr);
251 }
```

```
1 /*
2  * emb/Types.h
3  *
4  * type definitions for the drl quad-rotor project
5  *
6  * Brian J. Julian
7  *
8  * bjulian{at}mit{dot}edu
9  *
10 * Version 0.1
11 *
12 * 31 March 2009
13 *
14 */
15
16
17 // __EMB_TYPES_H__
18
19 #ifndef __EMB_TYPES_H__
20 #define __EMB_TYPES_H__
21
22
23 // position type
24
25 #ifndef pos_t
26 #define pos_t int32_t
27 #endif
28
29
30 // AutoPilot command input type
```

```
31
32 #ifndef cmd_t
33 #define cmd_t int16_t
34 #endif
35
36
37 // robot id type
38
39 #ifndef robotId_t
40 #define robotId_t uint32_t
41 #endif
42
43
44 // clock type
45
46 #ifndef clock_t
47 #define clock_t uint32_t
48 #endif
49
50
51 // robot name type
52
53 #ifndef robotName_t
54 #define robotName_t byte_t*
55 #endif
56
57
58 // crc16 type
59
60 #ifndef crc16_t
```

```
61 #define crc16_t uint16_t
62 #endif
63
64
65 // crc32 type
66
67 #ifndef crc32_t
68 #define crc32_t uint32_t
69 #endif
70
71
72 // __EMB_TYPES_H__
73
74 #endif
```

```

1  /*
2   * lpc/Buf.h
3   *
4   * buffer include for lpc214x
5   *
6   * Brian J. Julian
7   *
8   * bjulian{at}mit{dot}edu
9   *
10  * Version 0.1
11  *
12  * 31 March 2009
13  *
14  */
15
16
17 // __LPC_BUF_H__
18 #ifndef __LPC_BUF_H__
19 #define __LPC_BUF_H__
20
21
22 // includes
23
24 #include "lpc/Types.h"
25
26
27 // write buf macro
28
29 #define LPC_WRITEBUF(A,B) \
30     lpcWriteBuf(&A, (byte_t *)&B, sizeof(B))

```



```

31
32
33 // buffer struct
34
35 typedef struct
36 {
37     byte_t *Ptr;
38     int32_t WriteIndex;
39     int32_t ReadIndex;
40     int32_t Size;
41 }
42     lpcBuf_s;
43
44
45 // functions
46
47 int32_t lpcInitBuf(lpcBuf_s *Buf);
48 err32_t lpcEnableBuf(lpcBuf_s *Buf);
49
50 err32_t lpcAssignBuf(lpcBuf_s *Buf,
51                     byte_t *Ptr,
52                     int32_t Size);
53 err32_t lpcWriteBuf(lpcBuf_s *Buf,
54                    const byte_t *Write,
55                    const int32_t Size);
56 err32_t lpcReadBuf(lpcBuf_s *Buf,
57                   byte_t *Read,
58                   const int32_t Size,
59                   int32_t *BytesRecv);
60 err32_t lpcWriteByteBuf(lpcBuf_s *Buf,

```

```
61             const byte_t Write);
62 err32_t lpcReadByteBuf(lpcBuf_s *Buf,
63                       byte_t *Read,
64                       bool_t *ByteRecv);
65 err32_t lpcOffsetReadBuf(lpcBuf_s *Buf,
66                          int32_t Offset);
67
68
69 // --LPC-BUF-H--
70
71 #endif
```

```
1 /*
2  * lpc/Buf.c
3  *
4  * buffer source code for lpc214x
5  *
6  * Brian J. Julian
7  *
8  * bjulian{at}mit{dot}edu
9  *
10 * Version 0.1
11 *
12 * 31 March 2009
13 *
14 */
15
16
17 #include "lpc/Buf.h"
18
19 #include "lpc/Err.h"
20 #include "lpc/Types.h"
21
22
23
24 #define DEF_BUF_PTR (byte_t *)NULL
25 #define DEF_WRITE_INDEX 0
26 #define DEF_READ_INDEX 0
27
28
29
30 // initialize buffer
```

```

31 int32_t lpcInitBuf(lpcBuf_s *Buf)
32 {
33     // init buf structure
34     Buf->Ptr = DEF_BUF_PTR;
35     Buf->WriteIndex = DEF_WRITEINDEX;
36     Buf->ReadIndex = DEF_READINDEX;
37
38     // return successful
39     return(LPC_SUCC);
40 }
41
42
43
44 // enable buffer
45 err32_t lpcEnableBuf(lpcBuf_s *Buf)
46 {
47     // return successful
48     return(LPC_SUCC);
49 }
50
51
52
53 err32_t lpcAssignBuf(lpcBuf_s *Buf,
54                     byte_t *Ptr,
55                     int32_t Size)
56 {
57     Buf->Ptr = Ptr;
58     Buf->Size = Size;
59
60     // return successful

```

```

61     return(LPC.SUCC);
62 }
63
64
65
66
67 // read unread elements upto write index
68 err32_t lpcWriteBuf(lpcBuf_s *Buf,
69                    const byte_t *Write,
70                    const int32_t Size)
71 {
72     // local stack
73     int32_t ii;
74     int32_t BytesSent;
75
76     // check if valid size
77     if((Size < 0) || (Size > Buf->Size))
78     {
79         return(LPC.BUF_SIZE_ERR);
80     }
81
82     // do write
83     for(BytesSent = 0, ii = Buf->WriteIndex;
84         BytesSent < Size;
85         BytesSent++)
86     {
87         Buf->Ptr[ii++] = Write[BytesSent];
88         ii %= Buf->Size;
89     }
90

```

```

91  // update write index
92  Buf->WriteIndex = ii;
93
94  // return successful
95  return(LPC_SUCC);
96 }
97
98
99
100 // read unread elements upto write index
101 err32_t lpcReadBuf(lpcBuf_s *Buf,
102                  byte_t *Read,
103                  const int32_t Size,
104                  int32_t *BytesRecv)
105 {
106  // local stack
107  int32_t ii;
108
109  // check if valid size
110  if((Size < 0) || (Size > Buf->Size))
111  {
112      return(LPC_BUF_SIZE_ERR);
113  }
114
115  // do read
116  for(*BytesRecv = 0, ii = Buf->ReadIndex;
117      (*BytesRecv < Size) && (ii != Buf->WriteIndex);
118      (*BytesRecv)++)
119  {
120      Read[*BytesRecv] = Buf->Ptr[ii++];

```

```

121         ii %= Buf->Size ;
122     }
123
124     // update read index
125     Buf->ReadIndex = ii ;
126
127     // return successful
128     return(LPC.SUCC) ;
129 }
130
131
132
133 // read unread elements upto write index
134 err32_t lpcWriteByteBuf(lpcBuf_s *Buf,
135                         const byte_t Write)
136 {
137     Buf->Ptr[Buf->WriteIndex] = Write ;
138
139     // update write index
140     Buf->WriteIndex++ ;
141     Buf->WriteIndex %= Buf->Size ;
142
143     // return successful
144     return(LPC.SUCC) ;
145 }
146
147
148
149 // read unread elements upto write index
150 err32_t lpcReadByteBuf(lpcBuf_s *Buf,

```

```

151             byte_t *Read,
152             bool_t *ByteRecv)
153 {
154     // zero recv
155     *ByteRecv = false;
156
157     // do read
158     if (Buf->ReadIndex != Buf->WriteIndex)
159     {
160         *Read = Buf->Ptr[Buf->ReadIndex];
161
162         Buf->ReadIndex++;
163         Buf->ReadIndex %= Buf->Size;
164
165         *ByteRecv = true;
166     }
167
168     // return successful
169     return(LPC_SUCC);
170 }
171
172
173 // offset read index
174 err32_t lpcOffsetReadBuf(lpcBuf_s *Buf,
175                          int32_t Offset)
176 {
177     // check if valid size
178     if ((Offset < -Buf->Size) || (Offset > Buf->Size))
179     {
180         return(LPC_BUF_SIZE_ERR);

```



```
181     }
182
183     // offset read index
184     Buf->ReadIndex += Buf->Size;
185     Buf->ReadIndex += Offset;
186     Buf->ReadIndex %= Buf->Size;
187
188     // return successful
189     return(LPC_SUCC);
190 }
```

```
1 /*
2  * lpc/Cntl.h
3  *
4  * low level control include for lpc214x
5  *
6  * Brian J. Julian
7  *
8  * bjulian{at}mit{dot}edu
9  *
10 * Version 0.1
11 *
12 * 31 March 2009
13 *
14 */
15
16
17 // __LPC_CNTL_H__
18
19 #ifndef __LPC_CNTL_H__
20 #define __LPC_CNTL_H__
21
22
23 // includes
24
25 #include "lpc/Irq.h"
26 #include "lpc/Types.h"
27
28
29 // control defines
30
```

```

31 #define FOSC 14745600
32 #define MSEL 4
33 #define PSEL 2
34 #define CCLK (FOSC*MSEL)
35 #define FCCO (2*PSEL*CCLK)
36
37
38 // ctrl struct
39
40 typedef struct
41 {
42     lpcIrq_s Irq;
43 }
44     lpcCntl_s;
45
46
47 // function declarations
48
49 err32_t lpcInitCntl(lpcCntl_s *Cntl);
50 err32_t lpcEnableCntl(lpcCntl_s *Cntl);
51
52
53 // __LPC_CNTL_H__
54
55 #endif

```

```

1  /*
2  * lpc/Cntl.c
3  *
4  * low level control source code for lpc214x
5  *
6  * Brian J. Julian
7  *
8  * bjulian{at}mit{dot}edu
9  *
10 * Version 0.1
11 *
12 * 31 March 2009
13 *
14 */
15
16
17
18 #include "lpc/Cntl.h"
19
20 #include "lpc/Err.h"
21 #include "lpc/Irq.h"
22 #include "lpc/Lpc214x.h"
23 #include "lpc/Types.h"
24
25
26
27 // local defines
28 #define ENABLE_FIO_0 (1<<0)
29 #define ENABLE_FIO_1 (1<<1)
30 #define ENABLE_PLL (1<<0)

```

```

31 #define CONNECT_PLL (1<<1)
32 #define MASK_PLLLOCK_STAT (1<<10)
33 #define BUS_SAME_AS_CLK (1<<0)
34 #define DISABLEMAM 0
35 #define FETCH_CYCLE_6 ((1<<2)|(1<<1))
36 #define ENABLEMAM (1<<0)
37 #define PSEL_1 0
38 #define PSEL_2 (1<<5)
39 #define PSEL_4 (1<<6)
40 #define PSEL_8 ((1<<6)|(1<<5))
41
42
43
44 // local function declarations
45 void feedSeq(void);
46
47
48 // init low level controller
49 err32_t lpcInitCntl(lpcCntl_s *Cntl)
50 {
51     // set system control and status flags to high speed GPIO
52     SCS = ENABLE_FIO_0 | ENABLE_FIO_1;
53
54     // return success
55     return(LPC_SUCC);
56 }
57
58
59
60

```

```

61 // enable low level controller
62 err32_t lpcEnableCntl(lpcCntl_s *Cntl)
63 {
64     // configure the PLL
65     switch(PSEL)
66     {
67         case(2):
68             PLLCFG = (PSEL_2|(MSEL-1));
69             break;
70         case(4):
71             PLLCFG = (PSEL_4|(MSEL-1));
72             break;
73         case(8):
74             PLLCFG = (PSEL_8|(MSEL-1));
75             break;
76         default:
77             PLLCFG = (PSEL_1|(MSEL-1));
78             break;
79     }
80
81     // enable the PLL
82     PLLCON = ENABLE_PLL;
83
84     // perform correct feed sequence for PLL changes
85     feedSeq();
86
87     // wait for PLL status to change
88     while((PLLSTAT & MASK.PLLLOCK.STAT)==0);
89
90     // connect the PLL

```

```

91  PLLCON = ENABLE_PLL | CONNECT_PLL;
92
93  // perform correct feed sequence for PLL changes
94  feedSeq();
95
96  // set VPB to be same as processor clock speed
97  VPBDIV = BUS_SAME_AS_CLK;
98
99  // disable memory accelerator module
100 MAMCR = DISABLE_MAM;
101
102 // configure memory accelerator module timing control
103 MAMTIM = FETCH_CYCLE_6;
104
105 // enable memory accelerator module
106 MAMCR = ENABLE_MAM;
107
108 // initialize irq
109 lpcInitIrq(&Cntl->Irq);
110
111 // enable irq
112 lpcEnableIrq(&Cntl->Irq);
113
114 // return success
115 return(LPC_SUCC);
116 }
117
118
119
120 // perform correct feed sequence for PLL changes

```

```
121 void feedSeq(void)
122 {
123     PLLFEED = (0xAA);
124     PLLFEED = (0x55);
125 }
```



```
1 /*
2  * lpc/Eint.h
3  *
4  * external interrupt control source code for lpc214x
5  *
6  * Brian J. Julian
7  *
8  * bjulian{at}mit{dot}edu
9  *
10 * Version 0.1
11 *
12 * 31 March 2009
13 *
14 */
15
16
17 // __LPC_EINT_H__
18
19 #ifndef __LPC_EINT_H__
20 #define __LPC_EINT_H__
21
22
23 // includes
24
25 #include "lpc/Types.h"
26 #include "lpc/Vic.h"
27
28
29 // external interrupt struct
30
```

```

31 typedef struct
32 {
33     int32_t Num;
34     int32_t Pin;
35     reg32_t VectAddr;
36     lpcVic_s Vic;
37 }
38     lpcEint_s;
39
40
41 // function declarations
42
43 err32_t lpcInitEint(lpcEint_s *Eint);
44 err32_t lpcEnableEint(lpcEint_s *Eint);
45
46 err32_t lpcSetPinEint(lpcEint_s *Eint,
47                       const int32_t Pin);
48 err32_t lpcSetVectAddrEint(lpcEint_s *Eint,
49                             const reg32_t VectAddr);
50 err32_t lpcResetVicEint(const lpcEint_s *Eint);
51 err32_t lpcClearIrEint(const lpcEint_s *Eint);
52 err32_t lpcReadPinEint(const lpcEint_s *Eint,
53                       bool_t *Read);
54
55
56 // __LPC_EINT_H__
57
58 #endif

```

```
1 /*
2  * lpc/Eint.c
3  *
4  * external interrupt source code for lpc214x
5  *
6  * Brian J. Julian
7  *
8  * bjulian{at}mit{dot}edu
9  *
10 * Version 0.1
11 *
12 * 31 March 2009
13 *
14 */
15
16
17
18 #include "lpc/Eint.h"
19
20 #include "lpc/Err.h"
21 #include "lpc/Irq.h"
22 #include "lpc/Lpc214x.h"
23 #include "lpc/Types.h"
24
25
26
27 // local defines
28 #define DEF_PIN 0
29
30
```

```

31
32
33 // initialize external interrupt controller
34 err32_t lpcInitEint(lpcEint_s *Eint)
35 {
36     // assign vector address
37     Eint->VectAddr = (reg32_t)NULL;
38
39     // set default pin
40     Eint->Pin = DEF_PIN;
41
42     // initialize vector interrupt controller
43     lpcInitVic(&Eint->Vic);
44
45     // return success
46     return(LPC_SUCC);
47 }
48
49
50 // enable external interrupt controller
51 err32_t lpcEnableEint(lpcEint_s *Eint)
52 {
53     // check pin
54     switch(Eint->Pin)
55     {
56         // EINT0
57         case(1):
58         case(16):
59             Eint->Num = 0;
60             break;

```

```

61
62     // EINT1
63     case(3):
64     case(14):
65         Eint->Num = 1;
66         break;
67
68     // EINT2
69     case(7):
70     case(15):
71         Eint->Num = 2;
72         break;
73
74     // EINT3
75     case(9):
76     case(20):
77     case(30):
78         Eint->Num = 3;
79         break;
80
81     // error
82     default:
83         return(LPC_EINT_PIN_ERR);
84         break;
85     }
86
87     // configure pinsel
88     PINSEL0 &= ~((1<<(Eint->Pin<<1)) | (1<<((Eint->Pin<<1)+1)))
89     ;
89     switch(Eint->Pin)

```

```

90     {
91         // PINSEL 11
92         case(1):
93         case(3):
94         case(7):
95         case(9):
96             PINSEL0 |= (1<<(Eint->Pin<<1));
97             PINSEL0 |= (1<<(((Eint->Pin<<1)+1)));
98             break;
99
100        case(20):
101            PINSEL1 |= (1<<(((Eint->Pin-16)<<1)));
102            PINSEL1 |= (1<<(((Eint->Pin-16)<<1)+1)));
103            break;
104
105        // PINSEL 10
106        case(14):
107        case(15):
108            PINSEL0 &= ~(1<<(Eint->Pin<<1));
109            PINSEL0 |= (1<<(((Eint->Pin<<1)+1)));
110            break;
111
112        case(30):
113            PINSEL1 &= ~(1<<(((Eint->Pin-16)<<1)));
114            PINSEL1 |= (1<<(((Eint->Pin-16)<<1)+1)));
115            break;
116
117        // PINSEL 01
118        case(16):
119            PINSEL1 &= ~(1<<(((Eint->Pin-16)<<1)+1)));

```

```

120     PINSEL1 |= (1<<((Eint->Pin-16)<<1));
121     break;
122
123     // error
124     default:
125         return(LPC_EINT_PIN_ERR);
126         break;
127     }
128
129     // assign vic type
130     lpcSetTypeVic(&Eint->Vic,
131                 LPC_EINT_0_VIC + Eint->Num);
132
133     // assign vic vect addr
134     lpcSetVectAddrVic(&Eint->Vic,
135                     Eint->VectAddr);
136
137     // Set ext mode, polarity, and enable eint
138     EXTMODE |= (1<<(Eint->Num));
139
140     // enable eint vic
141     lpcEnableVic(&Eint->Vic);
142
143     // Clear interrupts
144     lpcClearIrEint(Eint);
145
146     // return success
147     return(LPC_SUCC);
148 }
149

```

```

150
151 err32_t lpcClearIrEint(const lpcEint_s *Eint)
152 {
153     // Clear interrupts
154     EXTINT = (1<<(Eint->Num));
155
156     // return success
157     return(LPC_SUCC);
158 }
159
160
161 err32_t lpcResetVicEint(const lpcEint_s *Eint)
162 {
163     VICVectAddr = LPC_CLR_REG;
164
165     // return success
166     return(LPC_SUCC);
167 }
168
169
170
171 err32_t lpcSetPinEint(lpcEint_s *Eint,
172                      const int32_t Pin)
173 {
174     Eint->Pin = Pin;
175
176     // return success
177     return(LPC_SUCC);
178 }
179

```



```

180
181
182
183 err32_t lpcSetVectAddrEint(lpcEint_s *Eint ,
184                             const reg32_t VectAddr)
185 {
186     Eint->VectAddr = VectAddr;
187
188     // return success
189     return(LPC.SUCC);
190 }
191
192
193
194 err32_t lpcReadPinEint(const lpcEint_s *Eint ,
195                       bool_t *Read)
196 {
197     // read pin state
198     *Read = (bool_t)((FIO0PIN & (1 << Eint->Pin)) != 0);
199
200     // return success
201     return(LPC.SUCC);
202 }

```

```

1  /*
2   * lpc/Err.h
3   *
4   * error definitions for lpc214x
5   *
6   * Brian J. Julian
7   *
8   * bjulian{at}mit{dot}edu
9   *
10  * Version 0.1
11  *
12  * 31 March 2009
13  *
14  */
15
16
17 // __LPC_ERR_H__
18
19 #ifndef __LPC_ERR_H__
20 #define __LPC_ERR_H__
21
22
23 // error definitions
24
25 #define LPC_SUCC 0
26 #define LPC_NUM_IRQ_SLOTS_ERR -1
27 #define LPC_BUF_SIZE_ERR -2
28 #define LPC_VECT_ADDR_ERR -3
29 #define LPC_VECT_CNTL_ERR -4
30 #define LPC_TMR_MATCH_IR_SLOT_ERR -5

```

```
31 #define LPC_UART_WRITE_ERR -6
32 #define LPC_UART_READ_ERR -7
33 #define LPC_EINT_PIN_ERR -7
34
35
36 // __LPC_ERR_H__
37
38 #endif
```

```

1  /*
2  * lpc/Irq.h
3  *
4  * interrupt request include for lpc214x
5  *
6  * Brian J. Julian
7  *
8  * bjulian{at}mit{dot}edu
9  *
10 * Version 0.1
11 *
12 * 31 March 2009
13 *
14 */
15
16
17 // __LPC_IRQ_H__
18
19 #ifndef __LPC_IRQ_H__
20 #define __LPC_IRQ_H__
21
22
23 // includes
24
25 #include "lpc/Types.h"
26
27
28 // irq struct
29
30 typedef struct

```

```
31 {
32     // dummy cpsr
33     unsigned Cpsr;
34 }
35     lpcIrq_s;
36
37
38 // function declarations
39
40 err32_t lpcInitIrq(lpcIrq_s *Irq);
41 err32_t lpcEnableIrq(lpcIrq_s *Irq);
42
43 err32_t lpcDisableIrq(lpcIrq_s *Irq);
44 err32_t lpcRestoreIrq(lpcIrq_s *Irq);
45
46
47 // __LPC_IRQ_H__
48
49 #endif
```

```

1 /*
2  * lpc/Irq.c
3  *
4  * interrupt source code for lpc214x
5  *
6  * Brian J. Julian
7  *
8  * bjulian{at}mit{dot}edu
9  *
10 * Version 0.1
11 *
12 * 31 March 2009
13 *
14 */
15
16
17 #include "lpc/Irq.h"
18
19 #include "lpc/Err.h"
20 #include "lpc/Lpc214x.h"
21 #include "lpc/Types.h"
22
23
24 static inline unsigned __get_cpsr(void);
25 static inline void __set_cpsr(unsigned Cpsr);
26 void defVectAddr(void) __attribute__((interrupt("IRQ")));
27
28
29 err32_t lpcDisableIrq(lpcIrq_s *Irq)
30 {

```

```

31     unsigned _cpsr;
32     _cpsr = __get_cpsr();
33     __set_cpsr(_cpsr | (1<<7));
34     return(LPC_SUCC);
35 }
36
37 err32_t lpcRestoreIrq(lpcIrq_s *Irq)
38 {
39     unsigned _cpsr;
40     _cpsr = __get_cpsr();
41     __set_cpsr((_cpsr & ~(1<<7)) | (Irq->Cpsr & (1<<7)));
42     Irq->Cpsr = _cpsr;
43     return(LPC_SUCC);
44 }
45
46 err32_t lpcEnableIrq(lpcIrq_s *Irq)
47 {
48     unsigned _cpsr;
49     _cpsr = __get_cpsr();
50     __set_cpsr(_cpsr & ~(1<<7));
51     Irq->Cpsr = _cpsr;
52     return(LPC_SUCC);
53 }
54
55 static inline unsigned __get_cpsr(void)
56 {
57     unsigned Cpsr;
58     asm volatile ("_mrs_/_%0,_cpsr" : "=r" (Cpsr) : /* no
        inputs */ );
59     return(Cpsr);

```

```

60 }
61
62 static inline void __set_cpsr(unsigned Cpsr)
63 {
64     asm volatile ("_msr_cpsr,_%0" : /* no outputs */ : "r" (
        Cpsr) );
65 }
66
67 void defVectAddr(void)
68 {
69     VICVectAddr = 0;
70     while(1);
71 }
72
73 err32_t lpcInitIrq(lpcIrq_s *Irq)
74 {
75     // Clear all IRQs and FIQs
76     VICIntEnClr = LPC_SET_REG;
77     VICIntSelect = LPC_CLR_REG;
78
79     // Set default interrupt handler
80     VICDefVectAddr = (reg32_t)defVectAddr;
81
82     return(LPC_SUCC);
83 }

```



```
1 /*
2  * lpc/Math.h
3  *
4  * verified math functions for lpc microcontrollers
5  *
6  * Brian J. Julian
7  *
8  * bjulian{at}mit{dot}edu
9  *
10 * Version 0.1
11 *
12 * 31 March 2009
13 *
14 */
15
16
17 // __LPC_MATH_H__
18
19 #ifndef __LPC_MATH_H__
20 #define __LPC_MATH_H__
21
22
23 // includes
24
25 #include "lpc/Types.h"
26
27
28 // mathematical constants
29
30 #define INT_PI 3216
```

```

31 #define INT_2_PI 6432
32 #define INT_PI_2 1608
33
34
35 // math macros
36
37 #define INT_MIN(A, B) (((A)<(B))?(A):(B))
38 #define INT_MAX(A, B) (((A)>(B))?(A):(B))
39 #define INT_MIN_MAX(A, B, C) (INT_MIN((B),INT_MAX((A),(C))))
40 #define INT_SIGN(A) (((A)>(0))?(1):(((A)<(0))?(−1):(0)))
41 #define INT_ABS(A) (((A)<(0))?(−A):(A))
42
43
44 // functions
45
46 int32_t int32Min(const int32_t *Array,
47                const int32_t Size);
48 int32_t int32Max(const int32_t *Array,
49                const int32_t Size);
50 int32_t int32Sum(const int32_t *Array,
51                const int32_t Size);
52 int32_t int32Dist(const int32_t *Pos1,
53                 const int32_t *Pos2,
54                 const int32_t Dim);
55 int32_t int32Sqrt(const int32_t Num);
56 int32_t int32Cos(int32_t Mrad);
57 int32_t int32Sin(int32_t Mrad);
58 int32_t int32Tan(const int32_t Mrad);
59 int32_t int32Atan(const int32_t Z);
60 int32_t int32Atan2(const int32_t Y,

```

```
61             const int32_t X);
62 int32_t int32Mrad(int32_t Mrad);
63 int32_t int32Pow(const int32_t Base,
64             const int32_t Exp);
65 int64_t int64Pow(const int64_t Base,
66             const int32_t Exp);
67 int64_t int64Sqrt(const int64_t Num);
68 int64_t int64Min(const int64_t *Array,
69             const int32_t Size);
70 int64_t int64Max(const int64_t *Array,
71             const int32_t Size);
72 int64_t int64Sum(const int64_t *Array,
73             const int32_t Size);
74
75
76 // --LPC_MATH_H--
77
78 #endif
```

```

1  /*
2   * lpc/Math.c
3   *
4   * verified math functions for lpc microcontrollers
5   *
6   * Brian J. Julian
7   *
8   * bjulian{at}mit{dot}edu
9   *
10  * Version 0.1
11  *
12  * 31 March 2009
13  *
14  */
15
16 // include files
17 #include "lpc/Math.h"
18
19 #include "lpc/Types.h"
20
21
22 // finds min value and corresponding index in an integer
   array
23 int32_t int32Min(const int32_t *Array,
24                const int32_t Size)
25 {
26     // local stack
27     int32_t ii;
28     int32_t Index;
29

```

```

30 // function
31 for(ii = 0, Index = 0;
32     ii < Size;
33     ii++)
34     {
35         if(Array[ii] < Array[Index])
36             {
37                 Index = ii;
38             }
39     }
40
41 return(Index);
42 }
43
44 // Finds max value and corresponding index in an integer
    array
45 int32_t int32Max(const int32_t *Array,
46                 const int32_t Size)
47 {
48     // local stack
49     int32_t ii;
50     int32_t Index;
51
52     // function
53     for(ii = 0, Index = 0;
54         ii < Size;
55         ii++)
56         {
57             if(Array[ii] > Array[Index])
58                 {

```

```

59         Index = ii;
60     }
61 }
62
63     return(Index);
64 }
65
66 // calculates sum of integer array
67 int32_t int32Sum(const int32_t *Array,
68                 const int32_t Size)
69 {
70     // local stack
71     int32_t ii;
72     int32_t Sum;
73
74     // function
75     for(ii = 0, Sum = 0;
76         ii < Size;
77         ii++)
78     {
79         Sum += Array[ii];
80     }
81
82     return(Sum);
83 }
84
85 // calculates distance between two position integer arrays
86 int32_t int32Dist(const int32_t *Pos1,
87                  const int32_t *Pos2,
88                  const int32_t Dim)

```

```

89 {
90     // local stack
91     int32_t ii;
92     int64_t Sum;
93
94     // function
95     for(ii = 0, Sum = 0;
96         ii < Dim;
97         ii++)
98     {
99         Sum += int64Pow((int64_t)(Pos1[ii]-Pos2[ii]),
100                        2);
101     }
102
103     return(int64Sqrt(Sum));
104 }
105
106 // calcuates integer square root of integer
107 int32_t int32Sqrt(const int32_t Num)
108 {
109     // local stack
110     int32_t Delta;
111     int32_t Pow;
112     int32_t Sqrt;
113
114     // function
115     for(Delta = (1<<15), Sqrt = 0;
116         Delta > 0;
117         Delta >>= 1)
118     {

```

```

119     Pow = int32Pow(Delta + Sqrt ,
120                   2);
121     if(Pow <= Num)
122     {
123         Sqrt += Delta;
124     }
125 }
126
127 return(Sqrt);
128 }
129
130 // calculates "milli"cos of milliradians
131 int32_t int32Cos(int32_t Mrad)
132 {
133     // local stack
134     int32_t C0, C2, C4;
135
136     // normalize milliradians
137     Mrad = int32Mrad(Mrad);
138
139     // function
140     if(INT_ABS(Mrad) > INT_PI_2)
141     {
142         return(-int32Cos(Mrad + INT_PI));
143     }
144
145     C0 = (1<<10);
146     C2 = (Mrad * Mrad) >> 10;
147     C4 = (Mrad * ((Mrad * C2) >> 10)) >> 10;
148

```



```

149     return(C0 - (C2/2) + (C4/24));
150 }
151
152 // Calculates "milli"sin of milliradians
153 int32_t int32Sin(int32_t Mrad)
154 {
155     // local stack
156     int32_t S1,S3,S5;
157
158     // normalize milliradians
159     Mrad = int32Mrad(Mrad);
160
161     // function
162     if(INT_ABS(Mrad) > INT_PI_2)
163     {
164         return(-int32Sin(Mrad + INT_PI));
165     }
166
167     S1 = Mrad;
168     S3 = (Mrad * ((Mrad * S1) >> 10)) >> 10;
169     S5 = (Mrad * ((Mrad * S3) >> 10)) >> 10;
170
171     return(S1 - (S3/6) + (S5/120));
172 }
173
174 // Calculates "milli"tan of milliradians
175 int32_t int32Tan(const int32_t Mrad)
176 {
177     return((int32Sin(Mrad)<<10)/int32Cos(Mrad));
178 }

```

```

179
180 // Calculates "milli" arctan of "milli" ratio
181 int32_t int32Atan(const int32_t Z)
182 {
183     // local stack
184     int32_t A1, A3, A5;
185
186     // function
187     if(INT_ABS(Z) > (1<<10))
188     {
189         return(INT_SIGN(Z)*INT_PI_2 - int32Atan((1 << 20) / Z))
190         ;
191     }
192
193     A1 = Z;
194     A3 = (Z * ((Z * A1) >> 10)) >> 10;
195     A5 = (Z * ((Z * A3) >> 10)) >> 10;
196
197     return(A1 - (A3/3) + (A5/5));
198 }
199
200
201 // Calculates "milli" arctan2 of "milli" ratio
202 int32_t int32Atan2(const int32_t Y,
203                  const int32_t X)
204 {
205     // local stack
206     int32_t Phi;
207     int32_t Atan2;

```

```

208
209 // function
210 switch(Y)
211 {
212     // Y == 0
213     case(0):
214         if(X < 0)
215         {
216             Atan2 = INT_PI;
217         }
218     else
219     {
220         Atan2 = 0;
221     }
222     break;
223
224     // Y != 0
225     default:
226         if(X == 0)
227         {
228             Atan2 = INT_PI_2*INT_SIGN(Y);
229         }
230     else
231     {
232         Phi = int32Pow(X,2);
233         Phi += int32Pow(Y,2);
234         Phi = int32Sqrt(Phi);
235         Phi += X;
236         Phi = (Y << 10) / Phi;
237         Atan2 = int32Atan(Phi);

```

```

238         Atan2 <<= 1;
239     }
240     break;
241 }
242
243 return(Atan2);
244 }
245
246 // normalizes milliradians into (INT_PI,INT_PI] range
247 int32_t int32Mrad(int32_t Mrad)
248 {
249     // function
250     while(INT_ABS(Mrad) > INT_PI)
251     {
252         Mrad -= INT_2_PI * INT_SIGN(Mrad);
253     }
254
255     return(Mrad);
256 }
257
258 // calculates integer power of an integer base
259 int32_t int32Pow(const int32_t Base,
260                const int32_t Exp)
261 {
262     // local stack
263     int32_t Pow;
264     int32_t ii;
265
266     // function
267     for(ii = 0, Pow = 1;

```

```

268     ii < Exp;
269     ii++)
270     {
271     Pow *= Base;
272     }
273
274     return(Pow);
275 }
276
277
278 // calculates integer power of an integer base
279 int64_t int64Pow(const int64_t Base,
280                const int32_t Exp)
281 {
282     // local stack
283     int32_t ii;
284     int64_t Pow;
285
286     // function
287     for(ii = 1, Pow = 1;
288         ii < Exp;
289         ii++)
290     {
291     Pow *= Base;
292     }
293
294     return(Pow);
295 }
296
297

```

```

298 // calculates integer square root of integer
299 int64_t int64Sqrt(const int64_t Num)
300 {
301     // local stack
302     int64_t Delta;
303     int64_t Pow;
304     int64_t Sqrt;
305
306     // function
307     for(Delta = (1<<31), Sqrt = 0;
308         Delta > 0;
309         Delta >>= 1)
310     {
311         Pow = int64Pow(Delta + Sqrt,
312                       2);
313         if(Pow <= Num)
314             {
315                 Sqrt += Delta;
316             }
317     }
318
319     return(Sqrt);
320 }
321
322
323 // finds min value and corresponding index in an integer
    array
324 int64_t int64Min(const int64_t *Array,
325                 const int32_t Size)
326 {

```

```

327 // local stack
328 int32_t ii;
329 int32_t Index;
330
331 // function
332 for(ii = 0, Index = 0;
333     ii < Size;
334     ii++)
335     {
336         if(Array[ii] < Array[Index])
337             {
338                 Index = ii;
339             }
340     }
341
342 return(Index);
343 }
344
345 // Finds max value and corresponding index in an integer
    array
346 int64_t int64Max(const int64_t *Array,
347                 const int32_t Size)
348 {
349     // local stack
350     int32_t ii;
351     int32_t Index;
352
353     // function
354     for(ii = 0, Index = 0;
355         ii < Size;

```

```

356     ii++)
357     {
358         if(Array[ ii ] > Array[Index])
359             {
360                 Index = ii;
361             }
362     }
363
364     return(Index);
365 }
366
367 // calculates sum of integer array
368 int64_t int64Sum(const int64_t *Array,
369                 const int32_t Size)
370 {
371     // local stack
372     int32_t ii;
373     int64_t Sum;
374
375     // function
376     for(ii = 0, Sum = 0;
377         ii < Size;
378         ii++)
379     {
380         Sum += Array[ ii ];
381     }
382
383     return(Sum);
384 }

```



```
1 /*
2  * lpc/Tmr.h
3  *
4  * timer include for lpc214x microcontrollers
5  *
6  * Brian J. Julian
7  *
8  * bjulian{at}mit{dot}edu
9  *
10 * Version 0.1
11 *
12 * 31 March 2009
13 *
14 */
15
16
17 // __LPC_TMR_H__
18
19 #ifndef __LPC_TMR_H__
20 #define __LPC_TMR_H__
21
22
23 // includes
24
25 #include "lpc/Vic.h"
26 #include "lpc/Types.h"
27
28
29 // defines
30
```

```

31 #define LPC_RESETTING_ON_TMR (1<<1)
32 #define LPC_RESETTING_OFF_TMR 0
33 #define LPC_STOPPING_ON_TMR (1<<2)
34 #define LPC_STOPPING_OFF_TMR 0
35
36
37 // timer struct
38
39 typedef struct
40 {
41     lpcVic_s Vic;
42     reg32_t VectAddr;
43     int32_t Num;
44     reg32_t BaseAddr;
45     int32_t ClosedIrSlots;
46 }
47 lpcTmr_s;
48
49
50 // match sub-structure for timer
51
52 typedef struct
53 {
54     lpcTmr_s *Tmr;
55     reg32_t Resetting;
56     reg32_t Stopping;
57     reg32_t IrSlot;
58     reg32_t BaseAddr;
59     reg32_t Tc;
60     int32_t Pin;

```

```

61 }
62     lpcMatchTmr_s;
63
64
65 // function
66
67 err32_t lpcInitTmr(lpcTmr_s *Tmr);
68 err32_t lpcEnableTmr(lpcTmr_s *Tmr);
69
70 err32_t lpcGetTcTmr(const lpcTmr_s *Tmr,
71                    reg32_t *Tc);
72 err32_t lpcInitMatchTmr(lpcMatchTmr_s *Match,
73                          lpcTmr_s *Tmr);
74 err32_t lpcEnableMatchTmr(lpcMatchTmr_s *Match);
75 err32_t lpcResetMatchTmr(lpcMatchTmr_s *Match,
76                           const reg32_t TcOffset);
77 err32_t lpcSetTcMatchTmr(lpcMatchTmr_s *Match,
78                           const reg32_t Tc);
79 err32_t lpcSetSlotMatchTmr(lpcMatchTmr_s *Match,
80                             const int32_t Slot);
81 err32_t lpcClrIrMatchTmr(lpcMatchTmr_s *Match);
82 err32_t lpcReadIrMatchTmr(lpcMatchTmr_s *Match,
83                             bool_t *Read);
84 err32_t lpcResetVicTmr(const lpcTmr_s *Tmr);
85 err32_t lpcSetNumTmr(lpcTmr_s *Tmr,
86                      const int32_t Num);
87 err32_t lpcSetVectAddrTmr(lpcTmr_s *Tmr,
88                            const reg32_t VectAddr);
89 err32_t lpcSetExtMatchTmr(const lpcMatchTmr_s *Match);
90 err32_t lpcSetPinMatchTmr(lpcMatchTmr_s *Match,

```

```
91             const int32_t Pin);
92 err32_t lpcReadPinMatchTmr(const lpcMatchTmr_s *Match,
93             bool_t *Read);
94 err32_t lpcSleepMsecTmr(const lpcTmr_s *Tmr,
95             uint32_t Msec);
96
97
98 // --LPC_TMR_H--
99
100 #endif
```

```
1 /*
2  * lpc/Tmr.c
3  *
4  * timer source code for lpc214x
5  *
6  * Brian J. Julian
7  *
8  * bjulian{at}mit{dot}edu
9  *
10 * Version 0.1
11 *
12 * 31 March 2009
13 *
14 */
15
16
17
18 #include "lpc/Tmr.h"
19
20 #include "lpc/Cntl.h"
21 #include "lpc/Err.h"
22 #include "lpc/Lpc214x.h"
23 #include "lpc/Types.h"
24 #include "lpc/Vic.h"
25
26
27
28 // defines
29 #define ENABLE_COUNTER (1<<0)
30 #define RESET_COUNTER (1<<1)
```

```

31 #define MIN_MATCH_IR_SLOT 0
32 #define MAX_MATCH_IR_SLOT 3
33 #define CLR_MCR ((1<<2)|(1<<1)|(1<<0))
34 #define DEF_RESETTING LPC_RESETTING_OFF_TMR
35 #define DEF_STOPPING LPC_STOPPING_OFF_TMR
36 #define ENABLE_MCR (1<<0)
37 #define DEF_TMR 0
38
39
40 /* The PINSEL0 register controls the functions of the pins as
   per the
41 settings listed in Table 63. The direction control bit in
   the
42 IO0DIR register is effective only when the GPIO function
   is
43 selected for a pin. For other functions, direction is
   controlled
44 automatically. */
45
46 #define PIN_SEL(A) \
47 LPC_REG((A) + PINSEL_BASE_ADDR)
48
49
50
51
52
53 /* Interrupt Register. The IR can be written to clear
   interrupts. The
54 IR can be read to identify which of the eight possible
   interrupt

```

```

55     sources are pending. */
56
57 #define T_IR(A) \
58     LPC_REG((A) + 0x00)
59
60
61
62 /* Timer Control Register. The TCR is used to control the
63     timer counter
64     functions. The Timer Counter can be disabled or reset
65     through the
66     TCR */
67
68 #define T_TCR(A) \
69     LPC_REG((A) + 0x04)
70
71 /* Timer Counter. The 32-bit TC is incremented every PR+1
72     cycles of
73     PCLK. The TC is controlled through the TCR. */
74
75 #define T_TC(A) \
76     LPC_REG((A) + 0x08)
77
78 /* Prescale Register. The Prescale Counter (below) is R/W
79     equal to
80     this value, the next clock increments the TC and clears
81     the PC. */

```

```

80
81 #define T_PR(A) \
82     LPC_REG((A) + 0x0C)
83
84
85
86 /* Prescale Counter. The 32-bit PC is a counter which is
      incremented
87     to the value stored in PR. When the value in PR is reached
      , the TC
88     is incremented and the PC is cleared. The PC is observable
      and
89     controllable through the bus interface. */
90
91 #define T_PC(A) \
92     LPC_REG((A) + 0x10)
93
94
95
96 /* Match Control Register. The MCR is used to control if an
      interrupt
97     is generated and if the TC is reset when a Match occurs.
      */
98
99 #define T_MCR(A) \
100     LPC_REG((A) + 0x14)
101
102
103

```



```

104 /* Match Register 0. MR0 can be enabled through the MCR to
      reset the
105     TC, stop both the TC and PC, and/or generate an interrupt
      every
106     time MR0 matches the TC. */
107
108 #define TMR(A) \
109     LPC_REG((A) + 0x18)
110
111
112 /* The External Match Register provides both control and
      status of the
113     external match pins MAT(0-3). */
114
115 #define TEMR(A) \
116     LPC_REG((A) + 0x3C)
117
118
119
120
121 // initialize timer
122 err32_t lpcInitTmr(lpcTmr_s *Tmr)
123 {
124     // assign vector address
125     Tmr->VectAddr = (reg32_t)NULL;
126
127     // default timer number
128     Tmr->Num = DEF_TMR;
129
130     // default interrupt register int

```

```

131  Tmr->ClosedIrSlots = LPC_CLR_REG;
132
133  // initialize vector interrupt controller
134  lpcInitVic(&Tmr->Vic);
135
136  // return success
137  return(LPC_SUCC);
138 }
139
140
141
142
143 // enable timer
144 err32_t lpcEnableTmr(lpcTmr_s *Tmr)
145 {
146  // assign vic type
147  lpcSetTypeVic(&Tmr->Vic,
148              LPC_TMR_0_VIC + Tmr->Num);
149
150  // assign vic vect addr
151  lpcSetVectAddrVic(&Tmr->Vic,
152                  Tmr->VectAddr);
153
154  // enable tmr vic
155  lpcEnableVic(&Tmr->Vic);
156
157  // calc base addr
158  Tmr->BaseAddr = TMR0_BASE_ADDR + Tmr->Num*(TMR1_BASE_ADDR -
159              TMR0_BASE_ADDR);

```

```

160 // clear interrupts
161 T_IR(Tmr->BaseAddr) = LPC_SET_REG;
162
163 // enable counters
164 T_TCR(Tmr->BaseAddr) = ENABLE_COUNTER;
165
166 // reset counters
167 T_TCR(Tmr->BaseAddr) |= RESET_COUNTER;
168 T_TCR(Tmr->BaseAddr) &= ~RESET_COUNTER;
169
170 // return success
171 return(LPC_SUCC);
172 }
173
174
175
176 // get current timer clock count
177 err32_t lpcGetTcTmr(const lpcTmr_s *Tmr,
178                    reg32_t *Tc)
179 {
180 // set TC
181 *Tc = T_TC(Tmr->BaseAddr);
182
183 // return success
184 return(LPC_SUCC);
185 }
186
187
188 // initialize match for timer
189 err32_t lpcInitMatchTmr(lpcMatchTmr_s *Match,

```

```

190             lpcTmr_s *Tmr)
191 {
192     // link timer structure
193     Match->Tmr = Tmr;
194
195     // set default resetting
196     Match->Resetting = DEF.RESETTING;
197
198
199     // set default stopping
200     Match->Stopping = DEF.STOPPING;
201
202     // return success
203     return(LPC.SUCC);
204 }
205
206
207
208 // enable match for timer
209 err32_t lpcSetSlotMatchTmr(lpcMatchTmr_s *Match,
210                            const int32_t Slot)
211 {
212     Match->IrSlot = Slot;
213
214     // return success
215     return(LPC.SUCC);
216 }
217
218
219

```

```

220 // enable match for timer
221 err32_t lpcEnableMatchTmr(lpcMatchTmr_s *Match)
222 {
223     // if no open match ir slot, return error
224     if((Match->Tmr->ClosedIrSlots & (1<<Match->IrSlot)) != 0)
225     {
226         return(LPC_TMR_MATCH_IR_SLOT_ERR);
227     }
228
229     // close selected ir slot
230     Match->Tmr->ClosedIrSlots |= (1<<Match->IrSlot);
231
232     // assign base address
233     Match->BaseAddr = TMR0_BASE_ADDR;
234     Match->BaseAddr += Match->Tmr->Num*(TMR1_BASE_ADDR -
        TMR0_BASE_ADDR);
235     Match->BaseAddr += Match->IrSlot*(0x04);
236
237     // set stopping and/or resetting
238     TMCR(Match->Tmr->BaseAddr) |= ((Match->Resetting | Match->
        Stopping)<<(3 * Match->IrSlot));
239
240     // return success
241     return(LPC_SUCC);
242 }
243
244
245
246 // enable match for timer
247 err32_t lpcResetMatchTmr(lpcMatchTmr_s *Match,

```

```

248             const reg32_t TcOffset)
249 {
250     // local stack
251     reg32_t Tc;
252
253     // reset match control register
254     TMCR(Match->Tmr->BaseAddr) &= ~(CLR_MCR << (3*Match->
        IrSlot));
255
256     // get current tc
257     lpcGetTcTmr(Match->Tmr,
258                &Tc);
259
260     // set match tc
261     lpcSetTcMatchTmr(Match,
262                      Tc + TcOffset);
263
264     // set match register
265     TMR(Match->BaseAddr) = Match->Tc;
266
267     // enable interrupts for match control register
268     TMCR(Match->Tmr->BaseAddr) |= (ENABLE_MCR << (3*Match->
        IrSlot));
269
270     // return success
271     return(LPC_SUCC);
272 }
273
274
275

```

```

276 // read current match pin state
277 err32_t lpcReadPinMatchTmr(const lpcMatchTmr_s *Match,
278                             bool_t *Read)
279 {
280     // read pin state
281     *Read = (bool_t)((FIO0PIN & (1 << Match->Pin)) != 0);
282
283     // return success
284     return(LPC_SUCC);
285 }
286
287
288
289
290 err32_t lpcSetPinMatchTmr(lpcMatchTmr_s *Match,
291                             const int32_t Pin)
292 {
293     // set pin
294     Match->Pin = Pin;
295
296     // scan through possible pins
297     switch(Pin)
298     {
299         // 0b01 enabled
300         case(19):
301         case(20):
302             PINSEL1 &= ~(1 << (((Pin-16)<<1)+1));
303             PINSEL1 |= (1 << ((Pin-16)<<1));
304             break;
305

```

```

306     // 0b10 enabled
307     case(3):
308     case(5):
309     case(12):
310     case(13):
311         PINSEL0 &= ~(1 << (Pin<<1));
312         PINSEL0 |= (1 << ((Pin<<1)+1));
313         break;
314
315     case(16):
316         PINSEL1 &= ~(1 << ((Pin-16)<<1));
317         PINSEL1 |= (1 << (((Pin-16)<<1)+1));
318         break;
319
320     // 0b11 enabled
321     case(17):
322     case(18):
323     case(22):
324     case(28):
325     case(29):
326         PINSEL1 |= (1 << ((Pin-16)<<1));
327         PINSEL1 |= (1 << (((Pin-16)<<1)+1));
328         break;
329
330     // else
331     default:
332         break;
333 }
334
335 // return success

```



```

336     return(LPC.SUCC);
337 }
338
339
340
341
342
343 err32_t lpcSetExtMatchTmr(const lpcMatchTmr_s *Match)
344 {
345     TEMR(Match->Tmr->BaseAddr) |= ((1<<Match->IrSlot) |
346                                     (1<<(4+2*Match->IrSlot)) |
347                                     (1<<(5+2*Match->IrSlot)));
348
349     // return success
350     return(LPC.SUCC);
351 }
352
353
354
355
356 // set match register for timer
357 err32_t lpcSetTcMatchTmr(lpcMatchTmr_s *Match,
358                          const reg32_t Tc)
359 {
360     // set register
361     Match->Tc = Tc;
362
363     // return success
364     return(LPC.SUCC);
365 }

```

```

366
367
368
369 // set match register for timer
370 err32_t lpcClrIrMatchTmr(lpcMatchTmr_s *Match)
371 {
372     // clear register
373     T_IR(Match->Tmr->BaseAddr) = (1<<Match->IrSlot);
374
375     // return success
376     return(LPC_SUCC);
377 }
378
379
380
381 // set match register for timer
382 err32_t lpcReadIrMatchTmr(lpcMatchTmr_s *Match,
383                          bool_t *Read)
384 {
385     // set register
386     *Read = (bool_t)((T_IR(Match->Tmr->BaseAddr) & (1<<Match->
387         IrSlot)) != 0);
388
389     // return success
390     return(LPC_SUCC);
391 }
392
393
394 err32_t lpcResetVicTmr(const lpcTmr_s *Tmr)

```

```

395 {
396     VICVectAddr = LPC_CLR_REG;
397
398     // return success
399     return(LPC_SUCC);
400 }
401
402
403 err32_t lpcSetNumTmr(lpcTmr_s *Tmr,
404                     const int32_t Num)
405 {
406     Tmr->Num = Num;
407
408     // return success
409     return(LPC_SUCC);
410 }
411
412
413
414 err32_t lpcSetVectAddrTmr(lpcTmr_s *Tmr,
415                           const reg32_t VectAddr)
416 {
417     Tmr->VectAddr = VectAddr;
418
419     // return success
420     return(LPC_SUCC);
421 }
422
423
424 err32_t lpcSleepMsecTmr(const lpcTmr_s *Tmr,

```

```

425             uint32_t Msec)
426 {
427     // local stack
428     reg32_t Tc;
429     reg32_t StartTc;
430
431     // convert Msec to counts
432     Msec *= (CCLK/1000);
433
434     // get start tc
435     lpcGetTcTmr(Tmr,
436                &StartTc);
437
438     // sleep
439     do
440     {
441         lpcGetTcTmr(Tmr,
442                    &Tc);
443         Tc -= StartTc;
444     }
445     while(Tc < Msec);
446
447     // return success
448     return(LPC_SUCC);
449 }

```

```

1  /*
2  * lpc/Types.h
3  *
4  * variable definitions for lpc214x
5  *
6  * Brian J. Julian
7  *
8  * bjulian{at}mit{dot}edu
9  *
10 * Version 0.1
11 *
12 * 31 March 2009
13 *
14 */
15
16
17 // __LPC_TYPES_H__
18
19 #ifndef __LPC_TYPES_H__
20 #define __LPC_TYPES_H__
21
22
23 // defines to modify registers
24
25 #define LPC_REG(ADDR) (*(volatile unsigned long *) (ADDR))
26 #define LPC_CLR_REG 0x00000000
27 #define LPC_SET_REG 0xffffffff
28
29
30 // null type

```

```
31
32 #ifndef NULL
33 #define NULL (void *)0x00
34 #endif
35
36
37 // bool types
38
39 #ifndef false
40 #define false 0
41 #endif
42
43 #ifndef true
44 #define true 1
45 #endif
46
47
48 // 8-bit general byte type
49
50 #ifndef byte_t
51 #define byte_t unsigned char
52 #endif
53
54
55 // 8-bit general bool type
56
57 #ifndef bool_t
58 #define bool_t char
59 #endif
60
```

```
61
62 // 8-bit signed char type
63
64 #ifndef char8_t
65 #define char8_t char
66 #endif
67
68
69 // 32-bit register type
70
71 #ifndef reg32_t
72 #define reg32_t volatile unsigned long
73 #endif
74
75
76 // 32-bit error type
77
78 #ifndef err32_t
79 #define err32_t long
80 #endif
81
82
83 // 8-bit signed integer type
84
85 #ifndef int8_t
86 #define int8_t char
87 #endif
88
89
90 // 8-bit unsigned integer type
```

```
91
92 #ifndef uint8_t
93 #define uint8_t unsigned char
94 #endif
95
96
97 // 16-bit signed integer type
98
99 #ifndef int16_t
100 #define int16_t short
101 #endif
102
103
104 // 16-bit unsigned integer type
105
106 #ifndef uint16_t
107 #define uint16_t unsigned short
108 #endif
109
110
111 // 32-bit signed integer type
112
113 #ifndef int32_t
114 #define int32_t long
115 #endif
116
117
118 // 32-bit unsigned integer type
119
120 #ifndef uint32_t
```



```
121 #define uint32_t unsigned long
122 #endif
123
124
125 // 64-bit signed integer type
126
127 #ifndef int64_t
128 #define int64_t long long
129 #endif
130
131
132 // 64-bit unsigned integer type
133
134 #ifndef uint64_t
135 #define uint64_t unsigned long long
136 #endif
137
138
139 // __LPC_TYPES_H__
140
141 #endif
```

```
1 /*
2  * lpc/Uart.h
3  *
4  * uart include for lpc214x
5  *
6  * Brian J. Julian
7  *
8  * bjulian{at}mit{dot}edu
9  *
10 * Version 0.1
11 *
12 * 31 March 2009
13 *
14 */
15
16
17 // __LPC_UART_H__
18
19 #ifndef __LPC_UART_H__
20 #define __LPC_UART_H__
21
22
23 // includes
24
25 #include "lpc/Types.h"
26 #include "lpc/Buf.h"
27 #include "lpc/Vic.h"
28
29
30 // defines
```

```

31
32 #define LPC_RX_BUF_SIZE_UART 256
33 #define LPC_TX_BUF_SIZE_UART 256
34 #define LPC_FLOW_CNTL_ON_UART true
35 #define LPC_FLOW_CNTL_OFF_UART false
36
37
38 // macro to read buffer
39
40 #define LPC_READ_UART(A, B, C) \
41     lpcReadUart(&A, (byte_t *)&B, sizeof(B), &C)
42
43
44 // buffer struct
45
46 typedef struct
47 {
48     lpcBuf_s RxBuf;
49     lpcBuf_s TxBuf;
50
51     byte_t RxBufMem[LPC_RX_BUF_SIZE_UART];
52     byte_t TxBufMem[LPC_TX_BUF_SIZE_UART];
53
54     int32_t Baudrate;
55     bool_t FlowCntl;
56     reg32_t BaseAddr;
57
58     reg32_t VectAddr;
59     lpcVic_s Vic;
60     int32_t Num;

```

```

61 }
62     lpcUart_s;
63
64
65 // functions
66
67 err32_t lpcInitUart(lpcUart_s *Uart);
68 err32_t lpcEnableUart(lpcUart_s *Uart);
69
70 err32_t lpcSetVectAddrUart(lpcUart_s *Uart,
71                             const reg32_t VectAddr);
72 err32_t lpcSetNumUart(lpcUart_s *Uart,
73                       const int32_t Num);
74 err32_t lpcSetBaudRateUart(lpcUart_s *Uart,
75                             const int32_t BaudRate);
76 err32_t lpcSetFlowCntlUart(lpcUart_s *Uart,
77                             const bool_t FlowCntl);
78 err32_t lpcSendUart(lpcUart_s *Uart);
79 err32_t lpcRecvUart(lpcUart_s *Uart);
80 err32_t lpcWriteUart(lpcUart_s *Uart,
81                      const byte_t *Buf,
82                      const int32_t Size);
83 err32_t lpcReadUart(lpcUart_s *Uart,
84                    byte_t *Buf,
85                    const int32_t Size,
86                    int32_t *BytesRecv);
87 err32_t lpcReadByteUart(lpcUart_s *Uart,
88                         byte_t *Read,
89                         bool_t *ByteRecv);
90 err32_t lpcOffsetReadUart(lpcUart_s *Uart,

```

```
91             int32_t Offset);
92 err32_t lpcWriteBufToUart(lpcUart_s *Uart,
93             lpcBuf_s *Buf);
94
95
96 // __LPC_UART_H__
97
98 #endif
```

```
1 /*
2  * lpc/Uart.c
3  *
4  * uart source code for lpc214x
5  *
6  * Brian J. Julian
7  *
8  * bjulian{at}mit{dot}edu
9  *
10 * Version 0.1
11 *
12 * 31 March 2009
13 *
14 */
15
16 #include "lpc/Uart.h"
17
18 #include "lpc/Types.h"
19 #include "lpc/Cntl.h"
20 #include "lpc/Err.h"
21 #include "lpc/Lpc214x.h"
22
23
24 #define DEF_BAUDRATE 38400
25 #define DEF_UART_NUM 0
26 #define THRE_WRITE_SIZE 14
27
28
29
```

```

30 /* The U_RBR is the top byte of the UART Rx FIFO. The top
    byte of the
31 Rx FIFO contains the oldest character received and can be
    read via
32 the bus interface. The LSB (bit 0) represents the
    oldest received
33 data bit. If the character received is less than 8 bits,
    the unused
34 MSBs are padded with zeroes. */
35
36 #define U_RBR(A) \
37     LPC_REG((A) + 0x00)
38
39
40 /* The U_THR is the top byte of the UART0 TX FIFO. The top
    byte is
41 the newest character in the TX FIFO and can be written
    via the bus
42 interface. The LSB represents the first bit to transmit.
    */
43
44 #define U_THR(A) \
45     LPC_REG((A) + 0x00)
46
47
48 /* The UART Divisor Latch is part of the UART Fractional Baud
    Rate
49 Generator and holds the value used to divide the clock
    supplied by

```

```

50  the fractional prescaler in order to produce the baud rate
      clock,
51  which must be 16x the desired baud rate (Equation 1). The
      U_DLL and
52  U_DLM registers together form a 16 bit divisor where UODLL
      contains
53  the lower 8 bits of the divisor and UODLM contains the
      higher 8
54  bits of the divisor. A 0x0000 value is treated like a 0
      x0001 value
55  as division by zero is not allowed. The Divisor Latch
      Access Bit
56  (DLAB) in U_LCR must be one in order to access the UART
      Divisor
57  Latches. */
58
59  #define U_DLL(A)          \
60    LPC_REG((A) + 0x00)
61
62  #define U_DLM(A)          \
63    LPC_REG((A) + 0x04)
64
65
66  /* The U_IER is used to enable UART interrupt sources. */
67
68  #define U_IER(A)          \
69    LPC_REG((A) + 0x04)
70
71

```



```

72 /* The U_IIR provides a status code that denotes the priority
    and
73 source of a pending interrupt. The interrupts are frozen
    during an
74 U_IIR access. If an interrupt occurs during an U_IIR
    access, the
75 interrupt is recorded for the next U_IIR access. */
76
77 #define U_IIR(A) \
78     LPC_REG((A) + 0x08)
79
80
81 /* The U_FCR controls the operation of the UART Rx and TX
    FIFOs. */
82
83 #define U_FCR(A) \
84     LPC_REG((A) + 0x08)
85
86
87 /* The U_LCR determines the format of the data character that
    is to be
88 transmitted or received. */
89
90 #define U_LCR(A) \
91     LPC_REG((A) + 0x0C)
92
93
94 /* The U_LSR is a read-only register that provides status
    information
95 on the UART TX and RX blocks. */

```

```

96
97 #define ULSR(A) \
98     LPC_REG((A) + 0x14)
99
100
101
102 lpcUart_s *LinkedUart[2];
103
104
105 void uartVectAddr(void) __attribute__((interrupt("IRQ")));
106
107
108
109 err32_t lpcInitUart(lpcUart_s *Uart)
110 {
111     Uart->Baudrate = DEF_BAUDRATE;
112
113     Uart->Num = DEF_UART_NUM;
114
115     Uart->VectAddr = (reg32_t)uartVectAddr;
116
117     lpcInitBuf(&Uart->RxBuf);
118     lpcInitBuf(&Uart->TxBuf);
119     lpcInitVic(&Uart->Vic);
120
121     lpcAssignBuf(&Uart->RxBuf,
122                 Uart->RxBufMem,
123                 sizeof(Uart->RxBufMem));
124     lpcAssignBuf(&Uart->TxBuf,
125                 Uart->TxBufMem,

```

```

126         sizeof(Uart->TxBufMem));
127
128
129
130     // return successful
131     return(LPC_SUCC);
132 }
133
134
135 err32_t lpcEnableUart(lpcUart_s *Uart)
136 {
137     int32_t PinReg;
138     reg32_t DivReg;
139     reg32_t DummyReg;
140
141     Uart->BaseAddr = UART0_BASE_ADDR + Uart->Num*(
        UART1_BASE_ADDR - UART0_BASE_ADDR);
142
143     PinReg = Uart->Num<<4;
144
145     DivReg = (CCLK)/(Uart->Baudrate<<4);
146
147     LinkedUart[Uart->Num] = Uart;
148
149     lpcSetTypeVic(&Uart->Vic,
150                 LPC_UART0_VIC + Uart->Num);
151
152     lpcSetVectAddrUart(Uart,
153                        (reg32_t)uartVectAddr);
154

```

```

155 // Disable Uart interrupt enable (UIER)
156 UIER(Uart->BaseAddr) = LPC.CLR_REG;
157
158 // Clear UIIR, URBR, ULSR
159 DummyReg = U_IIR(Uart->BaseAddr);
160 DummyReg = U_RBR(Uart->BaseAddr);
161 DummyReg = U_LSR(Uart->BaseAddr);
162
163 // Set Register 0 for TX and RX
164 PINSEL0 &= ~(1<<(PinReg+3));
165 PINSEL0 |= (1<<(PinReg+2));
166 PINSEL0 &= ~(1<<(PinReg+1));
167 PINSEL0 |= (1<<(PinReg));
168
169 // Configure line control register (ULCR)
170 ULCR(Uart->BaseAddr) = (1<<7) | (1<<1) | (1<<0);
171
172 // 9.3.2 Uart0 Divisor Latch Registers (UDLL)
173 UDLL(Uart->BaseAddr) = DivReg & (0xFF);
174
175 // 9.3.2 Uart0 Divisor Latch Registers (UDLM)
176 UDLM(Uart->BaseAddr) = (DivReg >> 8) & (0xFF);
177
178 // 9.3.9 Uart0 Line Control Register (ULCR)
179 ULCR(Uart->BaseAddr) = (1<<1) | (1<<0);
180
181 // 9.3.8 Uart FIFO Control Register (UFCR)
182 UFCR(Uart->BaseAddr) = (1<<0);
183
184 // Enable Uart Vic and Buf

```

```

185   lpcEnableBuf(&Uart->RxBuf);
186   lpcEnableBuf(&Uart->TxBuf);
187   lpcEnableVic(&Uart->Vic);
188
189   // Enable Uart Interrupt Enable Register (UIER)
190   U_IER(Uart->BaseAddr) = (1<<1) | (1<<0);
191
192   return(LPC_SUCC);
193 }
194
195
196
197 err32_t lpcSetVectAddrUart(lpcUart_s *Uart,
198                             const reg32_t VectAddr)
199 {
200   lpcSetVectAddrVic(&Uart->Vic,
201                     VectAddr);
202
203   return(LPC_SUCC);
204 }
205
206
207
208 err32_t lpcSetNumUart(lpcUart_s *Uart,
209                       const int32_t Num)
210 {
211   Uart->Num = Num;
212
213   // return success
214   return(LPC_SUCC);

```

```

215 }
216
217
218
219 err32_t lpcSetBaudRateUart(lpcUart_s *Uart,
220                             const int32_t BaudRate)
221 {
222     Uart->Baudrate = BaudRate;
223
224     // return success
225     return(LPC_SUCC);
226 }
227
228
229
230 err32_t lpcSetFlowCntlUart(lpcUart_s *Uart,
231                             const bool_t FlowCntl)
232 {
233     Uart->FlowCntl = FlowCntl;
234
235     // return success
236     return(LPC_SUCC);
237 }
238
239
240
241 void uartVectAddr(void)
242 {
243     lpcUart_s *Uart;
244     byte_t DummyByte;

```

```

245  reg32_t UIIR;
246
247  Uart = LinkedUart[0];
248  UIIR = U_IIR(Uart->BaseAddr);
249
250  if( (UIIR & (1<<0)) != 0 )
251  {
252      Uart = LinkedUart[1];
253      UIIR = U_IIR(Uart->BaseAddr);
254  }
255
256  // Figure out which port caused interrupt
257  while( (UIIR & (1<<0)) == 0 )
258  {
259      // Check type of interrupt
260      switch( UIIR & ( (1<<3) | (1<<2) | (1<<1) ) )
261      {
262          // Case receive line status
263          case( (1<<2) | (1<<1) ):
264              DummyByte = U_LSR(Uart->BaseAddr);
265              break;
266
267          // Case received data available
268          case( (1<<2) ):
269              lpcRecvUart(Uart);
270              break;
271
272          // Case character time-out indicator
273          case( (1<<3) | (1<<2) ):
274              DummyByte = URBR(Uart->BaseAddr);

```

```

275         break;
276
277         // Case THRE interrupt
278         case( (1<<1) ):
279             lpcSendUart(Uart);
280             break;
281
282         default:
283             DummyByte = U_LSR(Uart->BaseAddr);
284             DummyByte = U_RBR(Uart->BaseAddr);
285             break;
286     }
287
288     UIIR = U_IIR(Uart->BaseAddr);
289 }
290
291 // Clear vector addr
292 VICVectAddr = 0;
293 }
294
295
296 err32_t lpcSendUart(lpcUart_s *Uart)
297 {
298     // local stack
299     int32_t ii;
300     bool_t RecvByte;
301     byte_t ScratchByte;
302
303     // if transmitter is empty
304     if((U_LSR(Uart->BaseAddr) & (1<<5)) != 0)

```



```

305     {
306         for( ii = 0;
307             ii < THRE_WRITE_SIZE;
308             ii++)
309         {
310             // read byte from tx uart buf
311             lpcReadByteBuf(&Uart->TxBuf,
312                            &ScratchByte ,
313                            &RecvByte);
314
315             // break if no available data in uart buf
316             if(!RecvByte)
317                 {
318                     break;
319                 }
320
321             // else write byte to transmitter
322             else
323                 {
324                     U_THR(Uart->BaseAddr) = ScratchByte;
325                 }
326         }
327     }
328
329     // return successful
330     return(LPC_SUCC);
331 }
332
333
334 err32_t lpcRecvUart(lpcUart_s *Uart)

```

```

335 {
336     // local stack
337     byte_t ScratchByte;
338
339     // while receiver data available
340     while((U_LSR(Uart->BaseAddr) & (1<<0)) != 0)
341     {
342         // receive byte from receiver
343         ScratchByte = U_RBR(Uart->BaseAddr);
344
345         // write byte to uart buf
346         lpcWriteByteBuf(&Uart->RxBuf,
347                         ScratchByte);
348     }
349
350     // return successful
351     return(LPC_SUCC);
352 }
353
354
355
356 err32_t lpcWriteUart(lpcUart_s *Uart,
357                     const byte_t *Buf,
358                     const int32_t Size)
359 {
360     // check if valid size
361     if((Size < 0) || (Size > Uart->TxBuf.Size))
362     {
363         return(LPC_UART_WRITE_ERR);
364     }

```

```

365
366 // write to uart buf
367 lpcWriteBuf(&Uart->TxBuf,
368             Buf,
369             Size);
370
371 // see if empty
372 lpcSendUart(Uart);
373
374 // return successful
375 return(LPC_SUCC);
376 }
377
378
379 err32_t lpcWriteBufToUart(lpcUart_s *Uart,
380                          lpcBuf_s *Buf)
381 {
382 // local stack
383 bool_t ByteRecv;
384 byte_t Read;
385
386 // write buf to uart buf
387 lpcReadByteBuf(Buf,
388                &Read,
389                &ByteRecv);
390
391 while(ByteRecv)
392 {
393     lpcWriteByteBuf(&Uart->TxBuf,
394                    Read);

```

```

395     lpcReadByteBuf( Buf,
396                     &Read,
397                     &ByteRecv );
398 }
399
400 // see if empty
401 lpcSendUart(Uart);
402
403 // return successful
404 return(LPC_SUCC);
405 }
406
407
408
409 err32_t lpcReadUart(lpcUart_s *Uart,
410                    byte_t *Buf,
411                    const int32_t Size,
412                    int32_t *BytesRecv)
413 {
414     // check if valid size
415     if((Size < 0) || (Size > Uart->RxBuf.Size))
416     {
417         return(LPC_UART_READ_ERR);
418     }
419
420     // read to uart buf
421     lpcReadBuf(&Uart->RxBuf,
422               Buf,
423               Size,
424               BytesRecv);

```

```

425
426 // return successful
427 return(LPC_SUCC);
428 }
429
430
431 err32_t lpcReadByteUart(lpcUart_s *Uart,
432                        byte_t *Read,
433                        bool_t *ByteRecv)
434 {
435 // read to uart buf
436 lpcReadByteBuf(&Uart->RxBuf,
437               Read,
438               ByteRecv);
439
440 // return successful
441 return(LPC_SUCC);
442 }
443
444
445 err32_t lpcOffsetReadUart(lpcUart_s *Uart,
446                          int32_t Offset)
447 {
448 // move read index back
449 lpcOffsetReadBuf(&Uart->RxBuf,
450                Offset);
451
452 // return successful
453 return(LPC_SUCC);
454 }

```

```
1 /*
2  * lpc/Vic.h
3  *
4  * vector interrupt control source code for lpc214x
5  *
6  * Brian J. Julian
7  *
8  * bjulian{at}mit{dot}edu
9  *
10 * Version 0.1
11 *
12 * 31 March 2009
13 *
14 */
15
16
17 // --LPC_VIC_H--
18
19 #ifndef --LPC_VIC_H--
20 #define --LPC_VIC_H--
21
22
23 // includes
24
25 #include "lpc/Types.h"
26
27
28 // defines
29
30 #define LPC_ARM_CORE0_VIC 2
```

```

31 #define LPC_ARM_CORE1_VIC 3
32 #define LPC_TMR_0_VIC 4
33 #define LPC_TMR_1_VIC 5
34 #define LPC_UART_0_VIC 6
35 #define LPC_UART_1_VIC 7
36 #define LPC_PWM_0_VIC 8
37 #define LPC_I2C_0_VIC 9
38 #define LPC_SPI_0_VIC 10
39 #define LPC_SPI_1_VIC 11
40 #define LPC_PLL_VIC 12
41 #define LPC_RTC_VIC 13
42 #define LPC_EINT_0_VIC 14
43 #define LPC_EINT_1_VIC 15
44 #define LPC_EINT_2_VIC 16
45 #define LPC_EINT_3_VIC 17
46 #define LPC_AD_0_VIC 18
47 #define LPC_I2C_1_VIC 19
48 #define LPC_BOD_VIC 20
49 #define LPC_AD_1_VIC 21
50 #define LPC_USB_VIC 22
51
52
53 // vic struct
54
55 typedef struct
56 {
57     reg32_t BaseAddr;
58     int32_t IrqSlot;
59     reg32_t VectAddr;
60     reg32_t Type;

```

```
61 }
62     lpcVic_s;
63
64
65 // function
66
67 err32_t lpcInitVic(lpcVic_s *Vic);
68 err32_t lpcEnableVic(lpcVic_s *Vic);
69
70 err32_t lpcSetVectAddrVic(lpcVic_s *Vic,
71                             const reg32_t VectAddr);
72 err32_t lpcSetTypeVic(lpcVic_s *Vic,
73                         const reg32_t Type);
74
75
76 // __LPC_VIC_H__
77
78 #endif
```



```
1 /*
2  * lpc/Vic.c
3  *
4  * vector interrupt controller source code for lpc214x
5  *
6  * Brian J. Julian
7  *
8  * bjulian{at}mit{dot}edu
9  *
10 * Version 0.1
11 *
12 * 31 March 2009
13 *
14 */
15
16 #include "lpc/Vic.h"
17
18 #include "lpc/Err.h"
19 #include "lpc/Lpc214x.h"
20 #include "lpc/Types.h"
21
22
23
24 // local defines
25 #define NUM_IRQ_SLOTS 16
26 #define ENABLE_VECT_CNTL (1<<5)
27 #define DEF_VIC_TYPE 0
28
29
30
```

```

31 /* Vector Address Register. When an IRQ interrupt occurs, the
    R/W IRQ
32 service routine can read this register and jump to the
    value
33 read. */
34
35 #define VIC_VECT_ADDR(A) \
36     LPC_REG((A) + 0x100)
37
38
39
40 /* Vector control 0 register. Vector Control Registers 0–15
    each R/W
41 control one of the 16 vectored IRQ slots. Slot 0 has the
    highest
42 priority and slot 15 the lowest. */
43
44 #define VIC_VECT_CNTL(A) \
45     LPC_REG((A) + 0x200)
46
47
48
49 // global register to keep track of closed irq slots
50 reg32_t ClosedIrqSlots = LPC_CLR_REG;
51
52
53 // initialize vic
54 err32_t lpcInitVic(lpcVic_s *Vic)
55 {
56     // set default vector address

```

```

57  Vic->VectAddr = (reg32_t)NULL;
58
59  // set default vector control
60  Vic->Type = DEF_VIC_TYPE;
61
62  // return success
63  return(LPC_SUCC);
64 }
65
66
67
68 // enable vic
69 err32_t lpcEnableVic(lpcVic_s *Vic)
70 {
71  // if not valid vector control type, return error
72  if(Vic->Type == DEF_VIC_TYPE)
73  {
74      return(LPC_VECT_CNTL_ERR);
75  }
76
77  // find open vectored irq slot
78  for(Vic->IrqSlot = 0;
79      (ClosedIrqSlots & (1<<Vic->IrqSlot)) != 0;
80      Vic->IrqSlot++);
81
82  // if no irq slot is open, return error
83  if(Vic->IrqSlot >= NUM_IRQ_SLOTS)
84  {
85      return(LPC_NUM_IRQ_SLOTS_ERR);
86  }

```

```

87
88 // close selected irq slot
89 ClosedIrqSlots |= (1<<Vic->IrqSlot);
90
91 // assign base register
92 Vic->BaseAddr = VIC_BASE_ADDR + Vic->IrqSlot*(0x04);
93
94 // assign vector address
95 VIC_VECT_ADDR(Vic->BaseAddr) = (reg32_t)Vic->VectAddr;
96
97 // enable vector control
98 VIC_VECT_CNTL(Vic->BaseAddr) = (Vic->Type |
    ENABLE_VECT_CNTL);
99
100 // enable interrupt request to contribute to irq
101 VICIntEnable = (1<<Vic->Type);
102
103 // return success
104 return(LPC_SUCC);
105 }
106
107
108
109
110 err32_t lpcSetVectAddrVic(lpcVic_s *Vic,
111                          const reg32_t VectAddr)
112 {
113     Vic->VectAddr = VectAddr;
114
115     // return success

```

```
116     return(LPC.SUCC);
117 }
118
119
120 err32_t lpcSetTypeVic(lpcVic_s *Vic,
121                       const reg32_t Type)
122 {
123     Vic->Type = Type;;
124
125     // return success
126     return(LPC.SUCC);
127 }
```



# Bibliography

- [1] Autopilot - ascending technologies gmbh.
- [2] Hummingbird - ascending technologies gmbh.
- [3] Starmac - stanford testbed of autonomous rotorcraft for multi-agent control.
- [4] M. Achtelik, A. Bachrach, R. He, S. Prentice, and N. Roy. Autonomous navigation and exploration of a quadrotor helicopter in gps-denied indoor environments. In *IARC First Symposium on Indoor Flight Issues*, 2009.
- [5] M. Achtelik, A. Bachrach, R. He, S. Prentice, and N. Roy. Stereo vision and laser odometry for autonomous helicopters in gps-denied indoor environments. In *Proceedings of the International Society for Optical Engineering*, 2009.
- [6] E. Altug, J. P. Ostrowski, and R. Mahony. Control of a quadrotor helicopter using visual feedback. In *Proceedings of the IEEE International Conference on Robotics and Automation*, volume 1, pages 72–77, 2002.
- [7] E. Altug, J. P. Ostrowski, and C. J. Taylor. Quadrotor control using dual camera visual feedback. In *Proceedings of the IEEE International Conference on Robotics and Automation*, volume 3, pages 4294–4299, September 2003.
- [8] A. Bachrach, R. He, and N. Roy. Autonomous flight in unstructured and unknown indoor environments. In *Robotics Science and Systems Workshop on Autonomous Flying Vehicles - Fundamentals and Applications*, 2009.
- [9] R. W. Beard, T. W. McLain, D. B. Nelson, D. Kingston, and D. Johanson. Decentralized cooperative aerial surveillance using fixed-wing miniature uavs. *Proceedings of the IEEE*, 94(7):1306–1324, July 2006.
- [10] J. Black and T. Ellis. Multi camera image tracking. *Image and Vision Computing*, 11:1256–1267, 2006.
- [11] Y. Bouktir, M. Haddad, and T. Chettibi. Trajectory planning for a quadrotor helicopter. In *Proceedings of the 16th Mediterranean Conference on Control and Automation*, pages 1258–1263, June 2008.
- [12] F. Bullo, J. Cortes, and S. Martinez. *Distributed Control of Robotic Networks*. June 2008. Manuscript preprint. Electronically available at <http://coordinationbook.info>.

- [13] Q. Cai and J. K. Aggarwal. Tracking human motion in structured environments using a distributed-camera system. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 21(12):1241–1247, 1999.
- [14] J. R. Clapper, J. J. Young, J. E. Cartwright, and J. G. Grimes. Unmanned systems roadmap 2007-2032. Technical report, Secretary of Defense, DoD, December 2007.
- [15] R. Collins, A. J. Lipton, H. Fujiyoshi, and T. Kanade. Algorithms for cooperative multisensor surveillance. *Proceedings of the IEEE*, 89(10):1456–1477, 2001.
- [16] J. Cortes, S. Martinez, T. Karatas, and F. Bullo. Coverage control for mobile sensing networks. *IEEE Transactions on Robotics and Automation*, 20(2):243–255, April 2004.
- [17] Y. Cui, K. A. Hua, and A. Phillips. Leveraging 1-hop neighborhood knowledge for efficient flooding in wireless ad hoc networks. In *Proceedings of the 24th IEEE International Performance, Computing, and Communications Conference*, pages 347–354, April 2005.
- [18] Z. Drezner. *Facility Location: A Survey of Applications and Methods*. Springer Series in Operations Research. Springer-Verlag, New York, 1995.
- [19] J. Escareno, S. Salazar-Cruz, and R. Lozano. Embedded control of a four-rotor uav. In *Proceedings of the American Control Conference*, 2006.
- [20] S. G. Fowers, D.-J. Lee, B. J. Tippetts, K. D. Lillywhite, A. W. Dennis, and J. K. Archibald. Vision aided stabilization and the development of a quad-rotor micro uav. In *Proceedings of the International Symposium on Computational Intelligence in Robotics and Automation*, pages 143–148, June 2007.
- [21] R. Franz, M. Milam, and J. Hauser. Applied receding horizon control of the caltech ducted fan. In *Proceedings of the American Control Conference*, 2002.
- [22] A. Ganguli, J. Cortes, and F. Bullo. Maximizing visibility in nonconvex polygons: Nonsmooth analysis and gradient algorithm design. In *Proceedings of the American Control Conference*, pages 792–797, Portland, OR, June 2005.
- [23] M. F. Godwin, S. C. Spry, and J. K. Hedrick. *A Distributed System for Collaboration and Control of UAV Groups: Experiments and Analysis*, volume 588 of *Lecture Notes in Economics and Mathematical Systems*, chapter 9, pages 139–156. Springer Berlin Heidelberg, March 2007.
- [24] D. Gurdan, J. Stumpf, M. Achtelik, K.-M. Doth, G. Hirzinger, and D. Rus. Energy-efficient autonomous four-rotor flying robot controlled at 1 khz. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 361–366, April 2007.



- [25] Z. J. Haas. A new routing protocol for the reconfigurable wireless networks. In *Proceedings of the IEEE 6th International Conference on Universal Personal Communications Record*, volume 2, pages 562–566, October 1997.
- [26] T. Hamel, R. Mahony, R. Lozano, and J. Ostrowski. Dynamic modelling and configuration stabilization for an x4-flyer. In *Proceedings of the 15th IFAC World Congress*, 2002.
- [27] J. Hauser, S. Sastry, and G. Meyer. Nonlinear control design for slightly non-minimum phase systems: application to v/stol aircraft. *Automatica*, 28(4):665–679, 1992.
- [28] R. He, S. Prentice, and N. Roy. Planning in information space for a quadrotor helicopter in a gps-denied environment. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 1814–1820, May 2008.
- [29] E. Hecht. *Optics*. Addison Wesley, Reading, MA, 3 edition, 1998.
- [30] G. M. Hoffmann, D. G. Rajnarayan, S. L. Waslander, D. Dostal, J. S. Jang, and C. J. Tomlin. The stanford testbed of autonomous rotorcraft for multi agent control (starmac). In *Proceedings of the 23rd Digital Avionics Systems Conference*, 2004.
- [31] G. M. Hoffmann and C. J. Tomlin. Decentralized cooperative collision avoidance for acceleration constrained vehicles. In *Proceedings of the 47th IEEE Conference on Decision and Control*, pages 4357–4363, December 2008.
- [32] G. M. Hoffmann and C. J. Tomlin. Autonomous multi-agent search and rescue: Algorithms and experiments. In *Robotics Science and Systems Workshop on Autonomous Flying Vehicles - Fundamentals and Applications*, 2009.
- [33] G. M. Hoffmann, S. L. Wasl, and C. J. Tomlin. Quadrotor helicopter flight dynamics and control: Theory and experiment. In *Proceedings of the AIAA Guidance, Navigation, and Control Conference*, August 2007.
- [34] O. Holland, J. Woods, R. DeNardi, and A. Clark. Beyond swarm intelligence: The ultraswarm. In *Proceedings of the IEEE Swarm Intelligence Symposium*, 2005.
- [35] J. P. How, B. Bethke, A. Frank, D. Dale, and J. Vian. Real-time indoor autonomous vehicle test environment. *IEEE Control Systems Magazine*, 28(2):51–64, April 2008.
- [36] J. P. How, E. King, and Y. Kuwata. Flight demonstrations of cooperative control for uav teams. In *Proceedings of the AIAA 3rd Unmanned Unlimited Technical Conference*, Chicago, IL, September 2004.

- [37] B. J. Julian, M. Schwager, M. Angermann, and D. Rus. Location-based algorithm for multi-hopping state estimates within a distributed robot team. In *Field and Service Robotics*, Cambridge, MA, July 2009.
- [38] C. Kemp. *Visual Control of a Miniature Quad-Rotor Helicopter*. Doctoral thesis, Churchill College, University of Cambridge, 2006.
- [39] S. M. Khan and M. Shah. Consistent labeling of tracked objects in multiple cameras with overlapping fields of view. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25(10):1355–1360, 2003.
- [40] Seung kook Yun, David Alan Hjelle, Hod Lipson, and Daniela Rus. Planning the reconfiguration of grounded truss structures with truss climbing robots that carry truss elements. In *Proceedings of the IEEE International Conference on Robotics and Automation*, Kobe, Japan, May 2009.
- [41] S. Martinez, J. Cortes, and F. Bullo. Motion coordination with distributed information. *IEEE Control Systems Magazine*, 27(4):75–88, 2007.
- [42] Meister, N. Frietsch, C. Ascher, and G. F. Trommer. Adaptive path planning for a vtol-uav. In *Proceedings of the IEEE/ION Position, Location, and Navigation Symposium*, pages 1252–1259, May 2008.
- [43] A. Mokhtari and A. Benallegue. Dynamic feedback controller of euler angles and wind parameters estimation for a quadrotor unmanned aerial vehicle. In *Proceedings of the IEEE International Conference on Robotics and Automation*, 2004.
- [44] S.-Y. Ni, Y.-C. Tseng, Y.-S. Chen, and J.-P. Sheu. The broadcast storm problem in a mobile ad hoc network. In *Proceedings of the 5th annual ACM/IEEE International Conference on Mobile Computing and Networking*, pages 151–162, 1999.
- [45] Philips. *Volume 1: LPC214x User Manual*, July 2006.
- [46] L. C. A. Pimenta, V. Kumar, R. C. Mesquita, and G. A. S. Pereira. Sensing and coverage for a network of heterogeneous robots. In *Proceedings of the IEEE Conference on Decision and Control*, Cancun, Mexico, December 2008.
- [47] P. Pounds, R. Mahony, J. Gresham, P. Corke, and J. Roberts. Towards dynamically-favourable quad-rotor aerial robots. In *Proceedings of the Australian Conference on Robotics and Automation*, 2004.
- [48] P. Pounds, R. Mahony, P. Hynes, and J. Roberts. Design of a four-rotor aerial robot. In *Proceedings of the Australian Conference on Robotics and Automation*, 2002.

- [49] R. Rao, V. Kumar, and C. J. Taylor. Planning and control of mobile robots in image space from overhead cameras. In *Proceedings of the International Conference on Robotics and Automation*, Barcelona, Spain, April 18-22 2005.
- [50] A. Ryan, X. Xiao, S. Rathinam, J. Tisdale, M. Zennaro, D. Caveney, R. Sengupta, and J. K. Hedrick. A modular software infrastructure for distributed control of collaborating uavs. In *Proceedings of the AIAA Guidance, Navigation, and Control Conference*, August 2006.
- [51] S. Salazar-Cruz, A. Palomino, and R. Lozano. Trajectory tracking for a four rotor mini-aircraft. In *Proceedings of the 44th IEEE Conference on Decision and Control*, pages 2505–2510, December 2005.
- [52] M. Schwager, B. J. Julian, and D. Rus. Optimal coverage for multiple hovering robots with downward facing cameras. In *Proceedings of the IEEE International Conference on Robotics and Automation*, 2009.
- [53] M. Schwager, D. Rus, and J. J. Slotine. Decentralized, adaptive coverage control for networked robots. *International Journal of Robotics Research*, 28(3):357–375, March 2009.
- [54] D. Shim, H. Chung, H. J. Kim, and S. Sastry. Autonomous exploration in unknown urban environments for unmanned aerial vehicles. In *Proceedings of the AIAA Guidance, Navigation and Control Conference*, 2005.
- [55] H. Shim, T. J. Koo, F. Hoffmann, and S. Sastry. A comprehensive study of control design for an autonomous helicopter. In *Proceedings of the 37th IEEE Conference on Decision and Control*, 1998.
- [56] M.-T. Sun and T.-H. Lai. Location aided broadcast in wireless ad hoc network systems. In *Proceedings of the IEEE Wireless Communications and Networking Conference*, volume 2, pages 597–602, March 2002.
- [57] J. W. R. Taylor. *Jane's Pocket Book of Remotely Piloted Vehicles: Robot Aircraft Today*. Collier Books, New York, third edition, 1977.
- [58] R. Teo, J. S. Jang, and C. J. Tomlin. Automated multiple uav flight: The stanford dragonfly uav program. In *Proceedings of the 43rd IEEE Conference on Decision and Control*, Paradise Island, Bahamas, 2004.
- [59] G. P. Tournier, M. Valenti, J. P. How, and E. Feron. Estimation and control of a quadrotor vehicle using monocular vision and moire patterns. In *Proceedings of the AIAA Guidance, Navigation and Control Conference*, August 2006.