PROJECT KINGFISHER

Team J Final Report

Eitan Babcock, Ryan Gibbs, Shu-Kai Lin, Samuel Wang

May 5, 2016

Abstract

This paper describes our solution to the problem of landing a helicopter on a ship deck. Landing on ships in open water is inherently difficult, as they are not stationary objects. A ship's deck is a moving target, which adds uncertainty to landing any flight, especially with a pilot's constrained field of view, inhibited by the aircraft itself. These factors make the landing portion of flight one of the most dangerous. An autonomous landing control technology is proposed that utilizes infrared beacons to determine the position and characterize the movement of the landing deck. The characterized movements will be used to determine a safe landing time. This will allow the onboard system to calculate a trajectory to land safely and quickly. The controller will then execute this planned trajectory to meet the landing deck at the calculated time and position. This technology will improve the safety and efficiency of the landing process, preventing human injury as well as saving money by reducing expenses from recovering crashed aircraft.

Table of Contents

1. Project Description ... 1
2. Use Case ... 1
3. System-level Requirements ... 3
4. Functional Architecture ... 4
   - 4.1. Visual Structure: ... 4
   - 4.2. Inputs ... 4
   - 4.3. Sense ... 4
   - 4.4. Localize ... 4
   - 4.5. Predict ... 5
   - 4.6. Plan Trajectory ... 5
   - 4.7. Follow Trajectory ... 5
   - 4.8. Send Landing Signal ... 5
   - 4.9. Outputs ... 5
5. System-level trade studies ... 6
   - 5.1. Vision Subsystem Sensors: ... 6
   - 5.2. Deck Markings: ... 7
   - 5.3. Trajectory Planning Trade Studies ... 7
   - 5.4. Prediction Trade Study ... 8
   - 5.5. Quadrotor Trade Study ... 8
   - 5.6. Localization Trade Study ... 9
6. Cyberphysical Architecture ... 10
   - 6.1. Visual Structure ... 10
   - 6.2. Single Board Computer ... 10
   - 6.3. Additional Boards ... 10
   - 6.4. Power Distribution ... 11
7. System Description/Evaluation ... 11
   - 7.1. Subsystem Descriptions ... 11
     - 7.1.1. Overall System ... 11
     - 7.1.2. Mock Ship Deck ... 12
     - 7.1.2.1. Camera Subsystem ... 13
     - 7.1.2.2. Vision Subsystem ... 13
     - 7.1.3. Localization Algorithm Subsystem ... 14

7.1.4. State Estimation Algorithm Subsystem 15
7.1.5. Prediction Algorithm Subsystem 16
7.1.1. Trajectory Generation Algorithm Subsystem 16
7.1.2. Trajectory Following Algorithm Subsystem 17
7.1.3. Flight Control Subsystem 18
7.1.4. Power Distribution Subsystem 18
7.1.5. Rotorcraft Modifications 18
7.2. Modeling, Analysis, and Testing 20
7.2.1. Fisheye Lens 20
7.2.2. Fisheye Camera Accuracy Verification: 21
7.2.3. Power Distribution System 21
7.3. Performance evaluation against the Spring Validation Experiment (SVE) 23
7.3.1. Evaluate with Functional and Non-Functional Requirements 23
7.3.2. Evaluate with SVE test plan 24
7.4. Strong Points 24
7.5. Weak Points 25
8. Project Management 26
8.1. Schedule 26
8.2. Budget 27
8.3. Risk Management 28
9. Conclusions 31
9.1. Key Fall semester lessons learned 31
9.2. Future Work 32
9.2.1. Control Improvements 32
9.2.2. Prediction 32
9.2.3. Known Bugs to Fix 32
10. References 33

1. Project Description

Project Kingfisher aims to develop a sensor suite and trajectory planning software for an autonomous rotorcraft to land on ships without the use of GPS or radio. Military operations at sea often utilize air vehicles traveling to and from ships. Aircraft carriers are large enough to allow landing and takeoff of fixed wing aircraft, but smaller ships are being used to support rotorcraft vehicles. With varying ship sizes and sea states, the pilot of a helicopter must be highly experienced to safely land the aircraft on the moving deck.

With the performance improvement of processors and the maturity of sensor technologies, unmanned aerial vehicles (UAVs) are becoming popular in a wide range of applications. These applications include military service, aerial photography, surveillance, environment mapping, cargo shipping, etc. [1]. Work has been shared between man and machine, and the working efficiency and capability has been increased since UAVs can access places humans cannot and can operate more precisely. However, without intelligence onboard the vehicles, UAVs can be very dangerous. A collision between a UAV and a landing base can cause significant loss and even human injuries.

Therefore, a technology that can autonomously land a rotorcraft on a ship deck by combining sensing and prediction is a potential solution for improving landing performance. Project Kingfisher will utilize a vision system suite carried by a small-scale quadrotor to demonstrate algorithms that can safely land a rotorcraft on a dynamically moving deck.

2. Use Case

Mr. Gman is a pilot in the Navy. His main mission as a helicopter pilot is to survey the environment of the battle zone. Every time he goes on a mission, he is worried about whether he will be able to come back to the ship safely because the helicopter landing environment can be very dangerous. Beside the threat from enemies, natural disturbances such as wind, lack of lighting, and rough seas can easily cause an accident.

The Navy utilizes many resources to reduce the risk of helicopter landing. Every possible effort has been made to solve the problem, including intense training for pilots and more stable vehicle platforms. The military has realized that the limitation of human capability is a key factor that causes accidents. Since environment surveying can be done by unmanned machines, one of the solutions is to apply the autonomous pilot system on the helicopter. After decades of cooperative research with the Robotics Institute at Carnegie Mellon University, a sensor suite and trajectory planning software are mature enough to provide autonomous landing for helicopters.

With this new technology, Mr. Gman no longer needs to operate the helicopter during landing. Instead, he flies the helicopter to the vicinity of the ship and pushes a button to activate landing mode. During landing mode, the controls are taken over by the landing autopilot. In this mode, the helicopter can sense the location of the ship deck from a far distance via cameras and infrared beacons without the use of GPS. Although the waves of the ocean heave the ship deck, the helicopter is able to predict the motion to determine the point in time when the deck will be in the safest landing position. The helicopter determines the speed and trajectory it must fly to meet the deck at that point in time. The helicopter meets the deck safely and smoothly, though the deck is still heaving from the ship riding the waves. The landing mode then deactivates, sending a signal to Mr. Gman that the autopilot has successfully landed the helicopter on the deck.

3. System-level Requirements

Functional Requirements		
User Need	Performance Requirement	Mandatory/Desired
Identify the deck within the environment	1 false positive deck identification allowable in 100 trials	Desired
	Rotorcraft shall sense position of deck relative to the rotorcraft: 1. Detection at 5m distance 2. 10cm + 4cm/m allowable error	Mandatory
Predict dynamics of the deck	Predict the deck within 4cm/m of current distance to the deck	Mandatory
Rotorcraft shall robustly follow a trajectory	Follow planned trajectory 99% of time, within 0.8cm/m of current distance to deck	Desired
Rotorcraft shall land on the landing zone of the deck.	Rotorcraft shall land within 50cm of center of the deck, as measured from the center of the rotorcraft	Mandatory
	Rotorcraft shall land on a dynamically moving deck	Mandatory
	Rotorcraft shall perform 8 successful landings over a 10 cycle lifetime.	Desired

Nonfunctional Requirements		
User Need	Constraint/Requirement	Required/Desired
System shall operate in EMCON conditions.	Operation of autonomous landing mode shall be possible without radio commands and in GPS-denied conditions.	Mandatory
System shall operate with minimal user input.	Autonomous landing sequence shall execute after a single user "go" command.	Mandatory
Rotorcraft shall land safely.	System shall operate without damage to rotorcraft or deck.	Mandatory
Rotorcraft shall operate in varying environmental conditions.	System shall operate through steady wind up to 5mph.	Desired
	System shall operate through gust wind up to 10mph.	Desired
	System shall operate from minimum 0.0001 lux to maximum 100,000 lux	Desired

4. Functional Architecture

4.1. Visual Structure:

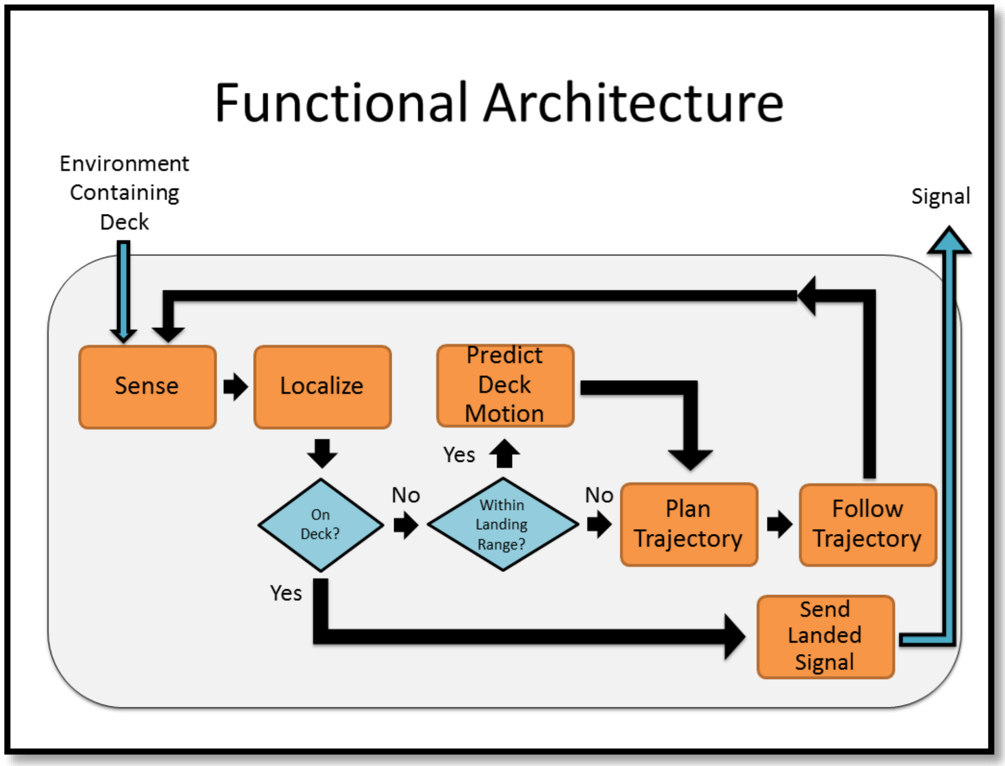

Figure 1: Functional Architecture of Landing Process

4.2. Inputs

The input to the autonomous landing system is an environment containing the ship deck. It should be noted that for our system, the assumption is made that the rotorcraft and the ship deck are the only objects in the environment besides the ground/water. We assume that there are no obstacles either on the deck or in the path of the rotorcraft.

4.3. Sense

The sensors output data about the environment at the maximum data rate that can be handled by the system.

4.4. Localize

In our system, the pilot will manually fly the rotorcraft to the proximity of the ship deck. Within landing range, a constructed frame is used to determine the relative position and orientation of the ship deck with respect to the rotorcraft.

4.5. Predict

In our system, we assume the motion of the ship deck is periodic. Using the history of deck poses, our system will predict the point in time when the deck is suitable for safe landing.

4.6. Plan Trajectory

Within landing distance, a trajectory is planned for the system to approach the deck. The goal point of the trajectory will be a safe hovering region above the ship deck. Within the safe hovering region, a second trajectory is planned for the system to land on the deck. This trajectory takes the predicted result into account.

4.7. Follow Trajectory

An open-loop command from trajectory system is fed from the system’s computer into the flight controller for the rotorcraft. A feedback controller in the system's computer will take the error between the trajectory and the current position of the system into account and produce the correction command to the flight controller.

4.8. Send Landing Signal

Upon landing, the system needs to send a confirmation signal to the user that it has landed. This allows the user to issue further commands or to perform some manual task on the rotorcraft.

4.9. Outputs

The system output is a landed rotorcraft and an output signal to the user that the system has landed successfully. The system will then enter an idle state awaiting further instructions, releasing autonomous control of the rotorcraft.

5. System-level trade studies

5.1. Vision Subsystem Sensors:

Category	Weight	Visible Camera	NIR/IR Camera	RaDAR	LiDAR	Ultrasonics
Field of View	5	5	5	2	10	4
Range	10	8	10	10	5	3
Payload	10	10	10	4	6	10
API/Support	20	10	10	6	10	8
Accuracy	20	8	8	6	10	6
Max Data Rate	5	10	10	5	7	5
Immunity to Weather Conditions	5	1	6	8	5	8
COST	10	9	8	3	2	10
ROS Development	10	10	10	2	8	8
Power	5	8	8	5	6	10
WEIGHTED TOTALS	1000	850	885	530	750	725

5.2. Deck Markings:

Category	Weight	No Markings	Fiducial at Center	Fiducials at corners	IR Beacons at Corners
Field of View Constraints	25	10	10	4	4
Visibility from Range	25	1	6	4	10
Ease of Detection	15	1	8	3	10
Useful Information Provided	25	1	10	8	8
Extent of Deck Modification	10	10	9	6	8
WEIGHTED TOTALS	1000	415	860	505	780

Infrared beacons were selected to be used at ranges too great to detect the center fiducial accurately. Once the fiducial is detectable, detection will switch to the center fiducial.

5.3. Trajectory Planning Trade Studies

Category	Weight	RRT*	Polynomial	RHC+Spline
Computational time	30	1	8	8
Aggressive flight	30	3	10	3
Optimization	10	10	7	8
Implementation	30	8	3	6
WEIGHTED TOTALS	1000	460	700	590

5.4. Prediction Trade Study

Category	Weight	Minor Component Analysis (MCA	Autoregressive (AR)	Neural Network (NN)	Wiener Prediction
Speed	25	8	10	5	10
Accuracy	25	10	8	6	8
Prediction Time Window	15	8	7	7	5
Ease of Implementation	35	6	10	4	5
WEIGHTED TOTALS	1000	780	905	520	700

5.5. Quadrotor Trade Study

Category	Weight	DJI Matrice 100	DJI Phantom	3DR X8+
Cost	5	1	5	7
Flight Time	5	7	7	3
API Support	9	5	5	10
Size	17	5	7	8
Community Support	10	5	5	10
Stability/Reliability	20	10	10	5
Wind Resistance	12	10	10	5
Maneuverability	10	10	10	10
Durability	10	10	7	5
WEIGHTED TOTALS	1000	760	784	706

5.6. Localization Trade Study

Category	Weight	Cloud Point Matching	Affine Transform/ Triangulation	Sensor combos using Particle Filtering
Accuracy	25	4	8	7
Reliability	25	4	8	4
Dynamic Capabilities	25	9	10	9
Supporting Documentation	10	8	10	10
Computational Load	15	1	10	6
WEIGHTED TOTALS	1000	520	900	690

6. Cyberphysical Architecture

6.1. Visual Structure

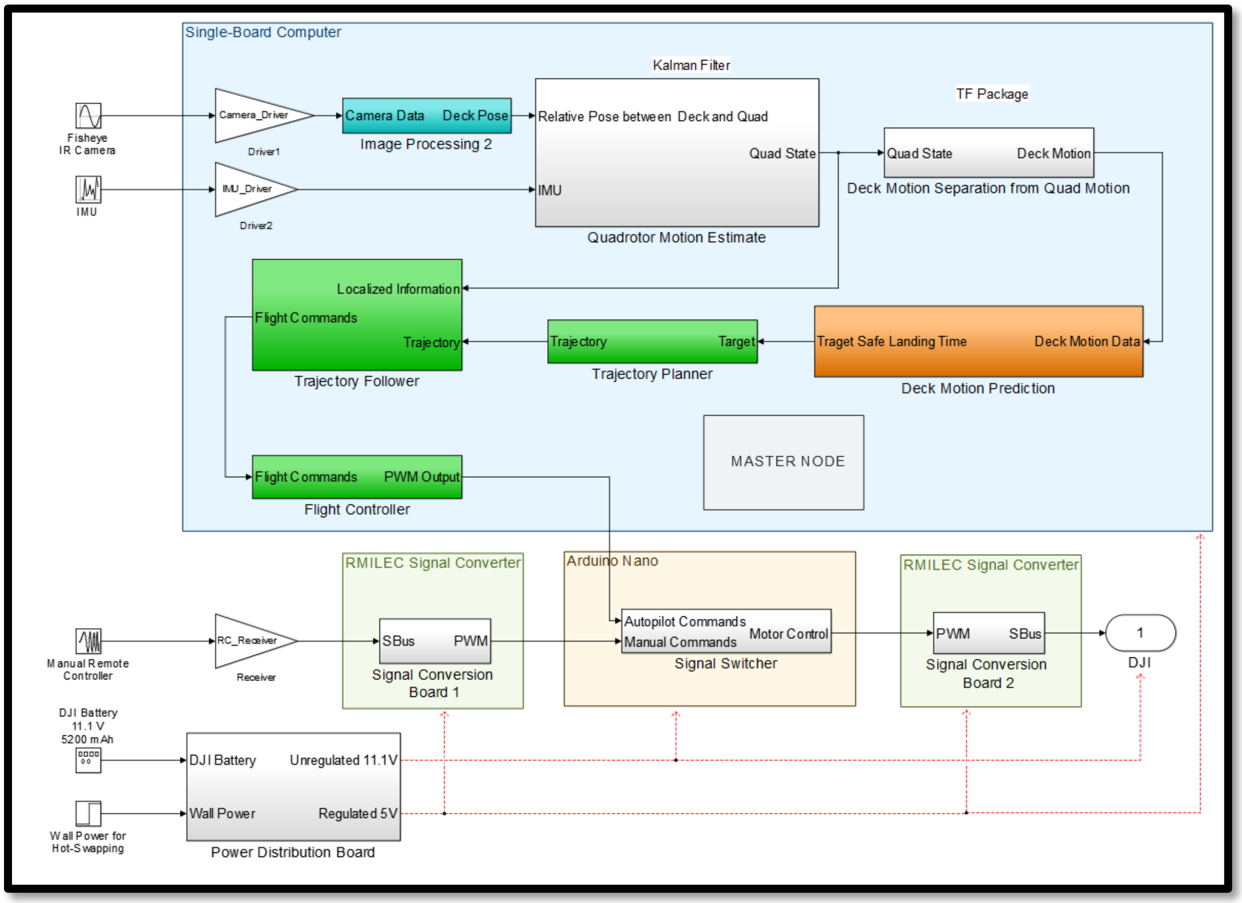

Figure 2: Cyberphysical Architecture of Landing System

6.2. Single Board Computer

Our single board computer houses the primary software and processing for our system. The image processing, localization, prediction, trajectory planning, trajectory following, and flight command subsystems all reside here. Additionally, there is a master logic node and master signal routing node that handles switching between states and routing the proper signals.

6.3. Additional Boards

In addition to our primary single board computer, we have two signal conversion boards and an Arduino Nano mounted on a custom printed circuit board. The signal conversion boards convert the remote control signal from SBus to PWM and the motor controls from PWM to SBus. This is needed because the DJI Phanom II expects SBus communication from the remote controller, but our single board computer and Arduino Nano use PWM communication. The Arduino Nano takes the inputs from the remote controller and the single board computer and determines which signal should be routed to the rotorcraft. The printed circuit board allows for all of these components to work together easily with no need for stray wires. It also allows us to flip a physical switch on the printed circuit board which bypasses the entire Arduino system and lets the quad connect directly to itself. This is useful for troubleshooting if the Arduino is not working.

6.4. Power Distribution

Our printed power distribution board takes power from the DJI Phantom II's battery and distributes it to the various other boards. There is a 5V regulator on this board to provide power to the single board computer and the conversion boards. The Arduino Nano has an internal power regulator, so it receives the full voltage of the battery. The cameras and IMU are powered via USB from the single board computer.

7. System Description/Evaluation

7.1. Subsystem Descriptions

7.1.1. Overall System

The system implemented in Project Kingfisher utilizes a combination of mechanical, electrical, and software subsystems which combine to form the finished working product. The figure below shows the final quadrotor sitting on our mock ship deck.

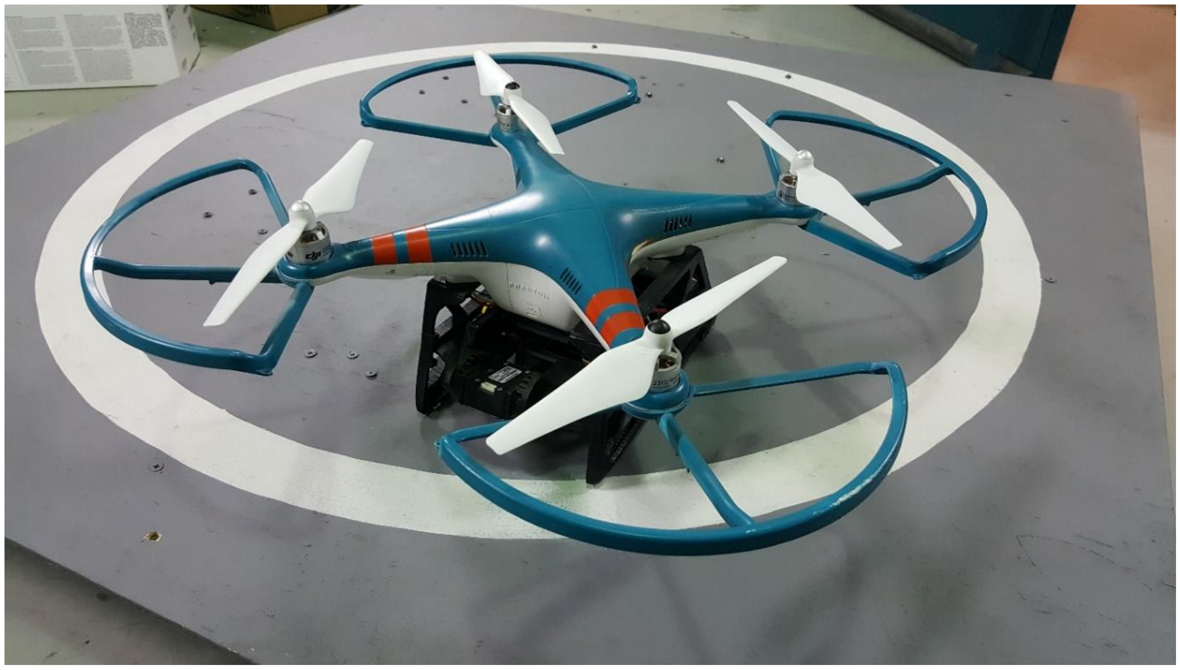

Figure 3: Final quadrotor on top of the mock ship deck

The quadrotor itself contains an onboard computer, camera, and an IMU. The software subsystems include a vision system, trajectory generation, trajectory following, prediction, and flight control. There is also a master node that handles all of the high level logic and signal routing.

7.1.2. Mock Ship Deck

The mock ship deck is designed to mimic a ship at sea in a controlled way. The mock ship deck has one actuated degree of freedom, as well as three manually controlled degrees of freedom. The actuated degree of freedom allows the deck to roll about its axis in a controlled way. The motion is periodic and mimics a ship at sea. The magnitude and frequency of the motion corresponds approximately to Sea State 4 or so – the Sea States are qualitatively determined, so it is hard to know exactly. The deck is mounted on lockable wheels so that it can be manually dragged along the ground, providing three degrees of freedom in the plane of the ground – horizontal translation in two directions, plus rotation about the normal.

There are also infrared LED beacons on the mock ship deck to help in the vision system. There are four beacons arranged in a square of 12" side length. Placing the beacons near the center of the deck provides a number of advantages. Centralized beacons maximize the blanking region surrounding the beacons. Since the deck itself is a highly controlled part of the environment seen by the quad, we can remove and severely limit the possibility of sources of interference on the deck itself. We arranged four infrared beacons in a square shape with known side length. A fifth beacon was placed at the center of the first four. This fifth beacon blinks with a square wave input at 2Hz.

In this configuration of beacons, the fifth blinking beacon lies at the intersection of the diagonals of the square formed by the first four beacons. Since straight lines are conserved, no matter how the shape shifts, warps, or rotates with perspective changes, the fifth beacon will always lie along both straight diagonal lines.

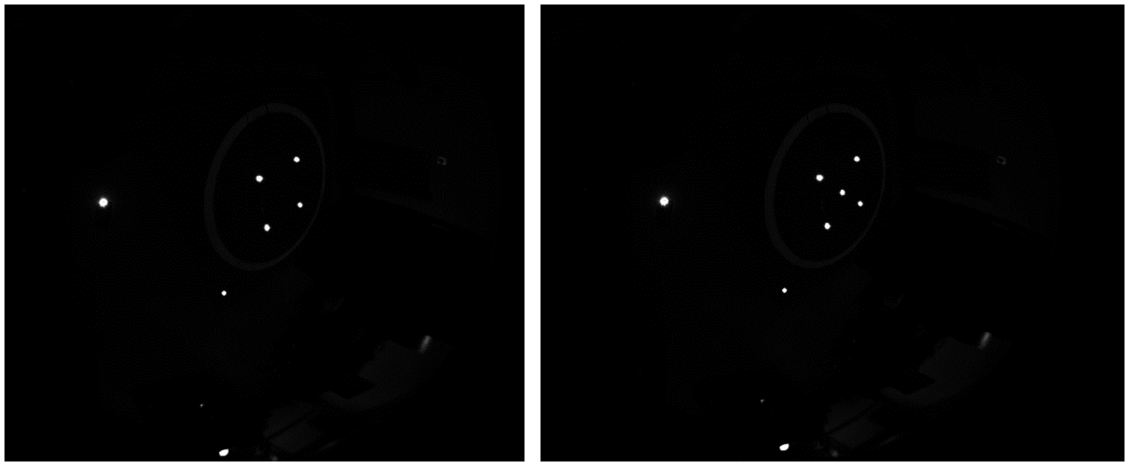
Figure 4: Raw image of the mock ship deck with the blinking beacon off (left) and on (right)

The actuators and beacons are powered by two on-board power supplies that sit on a shelf below the landing surface of the deck. A function generator is also on board to power the blinking LED. These three devices are plugged into a power strip so that only one power cord is required to run to the deck.

7.1.2.1. Camera Subsystem

A PointGrey Chameleon 3 camera was chosen as the camera for our system due to a few key features. High resolution at 1MP at a maximum framerate of 150 FPS gives us good control over exposure and enables the thresholding operation performed on the active beacons to be effective. Furthermore, the camera has been extensively modified, with the addition of a super-wide angle fisheye lens and modifications made to the physical lens mount holder to allow for a shorter back-flange distance from the lens to the sensor surface. The IR cut filter on the camera cover glass was also removed. The drivers for the camera needed to be compiled from source in order to be compatible with the ODroid single board computer. Further discussion of the fisheye lens and its calibration, rectification and use is presented in section 6.3.

7.1.2.2. Vision Subsystem

We implemented an object detection and tracking algorithm that is robust to multiple sources of error. Perspective changes, object motion, ego motion and lighting variations can all cause major problems for tracking algorithms. Our solution utilizes prior information of the target object’s 3D geometry in order to detect and track the deck with no prior knowledge of the starting relationship to the camera. The algorithm is efficient enough to run on an embedded system.

In order to detect the desired beacons in the camera’s field of view and distinguish them from other noisy light sources, we utilized the fact that we know the configuration of the beacons in 3D space and take advantage of the fact that straight lines are conserved with perspective transforms.

Our algorithm assumes a camera frame that contains at least four light sources. A connected components algorithm is run on the image to provide the locations in the image of each distinct light source. These light sources are iteratively grouped into two sets of paired potential beacons. The intersection of the lines formed by each pair of points is calculated to determine if there is the possibility of having a fifth beacon at that intersection.

Not every intersection between two lines can feasibly contain the fifth beacon that we are searching for. Several checks are done at this stage of the algorithm to rule out impossible cases so that only feasible cases are stored and checked in the next frame. When the blink is detected at one of these locations, we know that the parent beacons that created this intersection are the true deck beacons.

The tracking portion of algorithm is almost completely independent of the blink detection segment. For this portion of the algorithm, only the four steady beacons are considered – the fifth beacon is not needed.

Since the four beacons have a known configuration in 3D space, the Perspective-n-Point solution can be used to extract the relative 3D translation and rotation between the camera frame and the coordinate frame located at the center of the four beacons. We used OpenCV’s solvePnP function for this. This information is captured for two consecutive frames. The difference in position and orientation is calculated as the delta motion between these two frames. A constant velocity model is assumed, and so we can predict the 3D location of the camera relative to the beacons in the third frame prior to receiving a third frame.

The new 3D points are re-projected back into the camera plane. We then form a bounding box around each re-projected point, and only check within these bounding boxes for beacons in this third frame. This eliminates all noise from the image, as we are only looking for beacons where we predict there should be beacons. If the tracking algorithm fails, we return to the blink detection portion of the algorithm.

7.1.3. Localization Algorithm Subsystem

The input to the localization algorithm is the visual odometry from the vision subsystem and the inertial data from the rotorcraft IMU. Let landing range be the range within which the deck beacons are within the field of view of the camera.

Once within landing range, in addition to outputting the centroid location as above, the algorithm also reconstructs the absolute orientation of the deck. To do this, this algorithm builds a frame transform tree. The frames include a world in North, East, Down (NED) reference frame, quadrotor frame, camera frame, and the deck frame. The vision system provides the relationship between the ship deck and the camera. The IMU gives the motion between the quadrotor and the world frame. The camera is mounted on the quadrotor with fixed orientation and translation. With this setup, the transform tree provides the absolute motion of the ship deck in world frame. The motions of the quadrotor and the ship deck estimated from the vision system are thus separated.

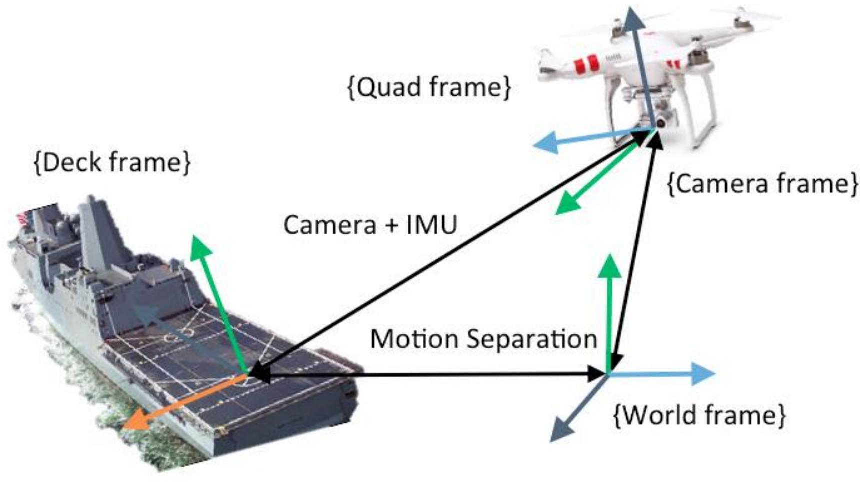

Figure 5: Coordinate frames involved in the localization subsystem

7.1.4. State Estimation Algorithm Subsystem

The state estimation fuses the visual odometry from vision subsystem and acceleration from IMU to provide more accurate and more responsive state estimation. Kalman filter is used as the framework for sensor fusion. The acceleration readings from IMU are used as prediction input to the process model at 130 Hz. The visual odometry is used as sensor update at 5 Hz.

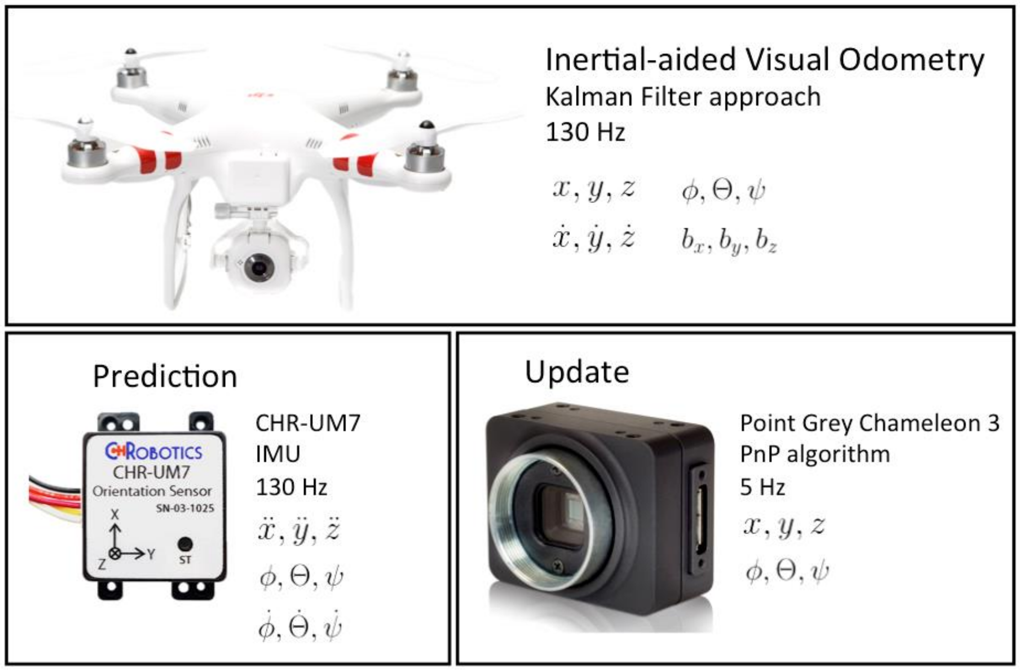
Inertial-aided Visual Odometry
Kalman Filter approach
130 Hz
x, y, z ϕ, Θ, ψ
$\dot{x}, \dot{y}, \dot{z}$ b_x, b_y, b_z
Prediction
CHR-UM7
IMU
130 Hz
$\ddot{x}, \ddot{y}, \ddot{z}$
ϕ, Θ, ψ
$\dot{\phi}, \dot{\Theta}, \dot{\psi}$
Update
Point Grey Chameleon 3
PnP algorithm
5 Hz
x, y, z
ϕ, Θ, ψ

Figure 6: Kalman filter state estimation using IMU and a camera

7.1.5. Prediction Algorithm Subsystem

In this project, only the roll of the deck is predicted. All other motion of the deck is treated as noise and handled by the controller. The roll of the deck is observed and recorded. This information is analyzed in real time to determine when safe landing times are occurring. A safe landing time is defined such that the deck is flat compared to gravity. The time difference between safe landing times is calculated, and this time difference is filtered and propagated forward.

Several assumptions are made about the motion of the deck. The motion is assumed to be periodic. The period of motion has to be relatively constant on the time scale of about 4 cycles. The motion is also assumed to be symmetric. That is to say that the deck rolls just as far to the right as it rolls to the left, and spends the same amount of time on each side.

7.1.1. Trajectory Generation Algorithm Subsystem

The trajectory generation system is responsible for generating a trajectory from initial state to a target position. When the user triggers the autonomous landing system, the trajectory generation subsystem will receive current pose of the quadrotor from the localization subsystem and sets the pose as initial position. Then the trajectory generation subsystem will generate a trajectory based on a quintic polynomial from the initial position to the pre-defined goal position. The figure below shows one of the quintic polynomial.

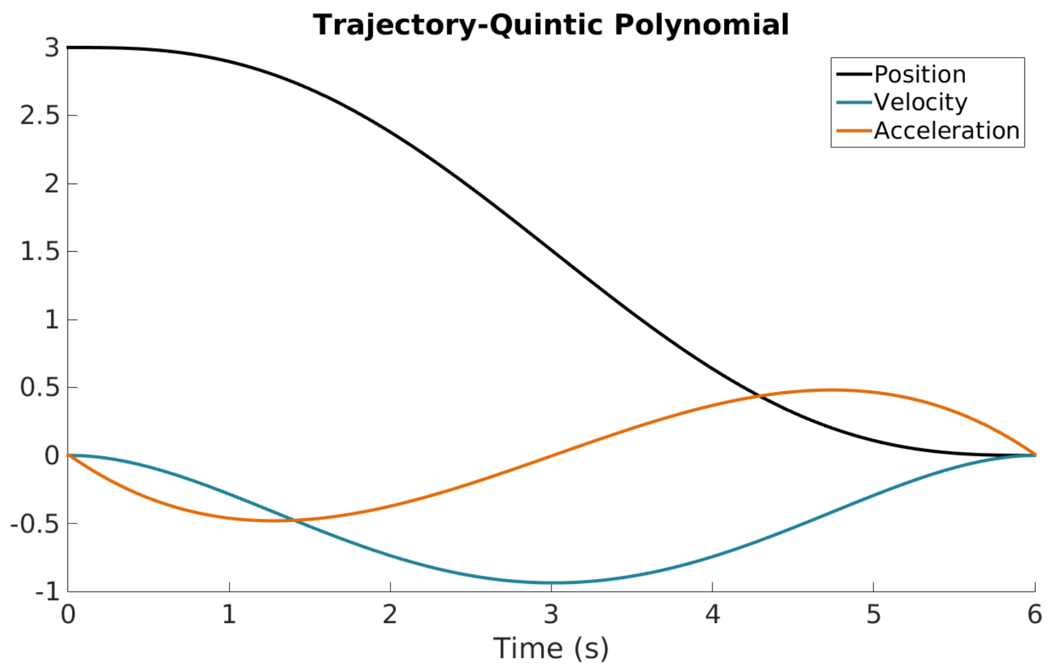

Figure 7: Quintic polynomial trajectory

7.1.2. Trajectory Following Algorithm Subsystem

The trajectory following algorithm subsystem is responsible for detecting errors in our current flightpath and making the corresponding course corrections to put the rotorcraft back on the target trajectory generated by the trajectory generation algorithm.

The figure below shows the block diagram of the trajectory following algorithm. The input command here is the position trajectory generated by the trajectory generation algorithm, and the sensor feedback is the feedback data from state estimation algorithm. We use PD controller as feedback controller and use acceleration command generated by quintic polynomial as a feedforward term. The advantage for using feedforward term is it can not only mitigate the following error but also decrease the amount of overshoot. Since there is an attitude controller controlled by DJI internally, we don't need to add additional inner loop to control the attitude.

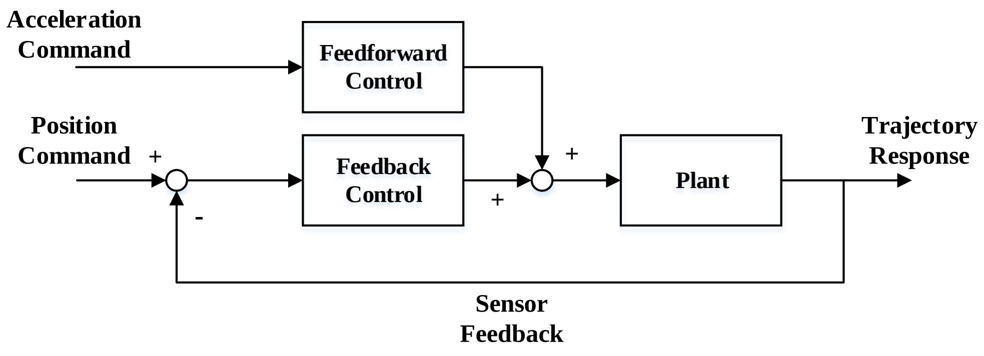

Figure 8: Block diagram of the trajectory following subsystem

The figure below shows one of the test data while the quadrotor is following the trajectory.

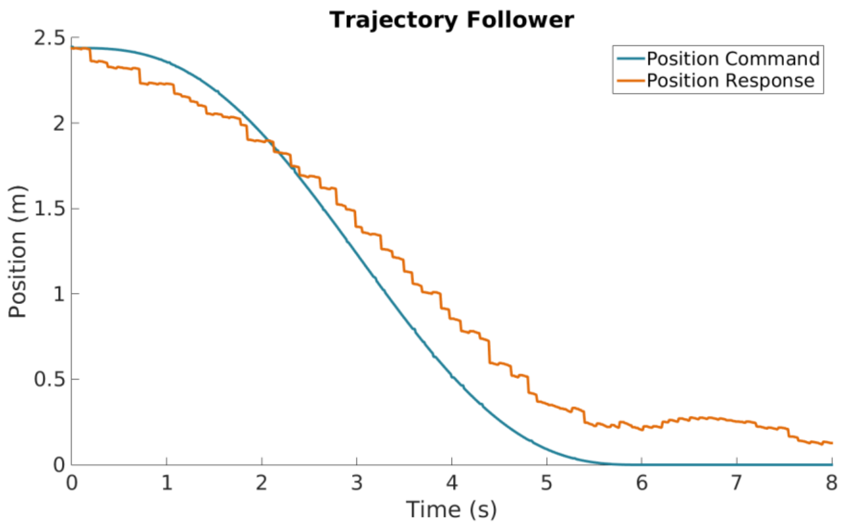

Figure 9: Experiment data of the trajectory following subsystem

7.1.3. Flight Control Subsystem

The flight control subsystem is responsible for converting the commands output from trajectory generation in terms of roll, pitch, yaw, and thrust to the necessary stick commands to emulate to the flight controller onboard the rotorcraft. Additionally, this subsystem toggles between manual flight of the rotorcraft by remote control and autonomous flight. Autonomous flight is achieved by emulating manual flight commands from the "remote control" generated by the trajectory generation and trajectory following algorithms. Control can be returned to the human safety pilot through the use of an extra channel on the remote control.

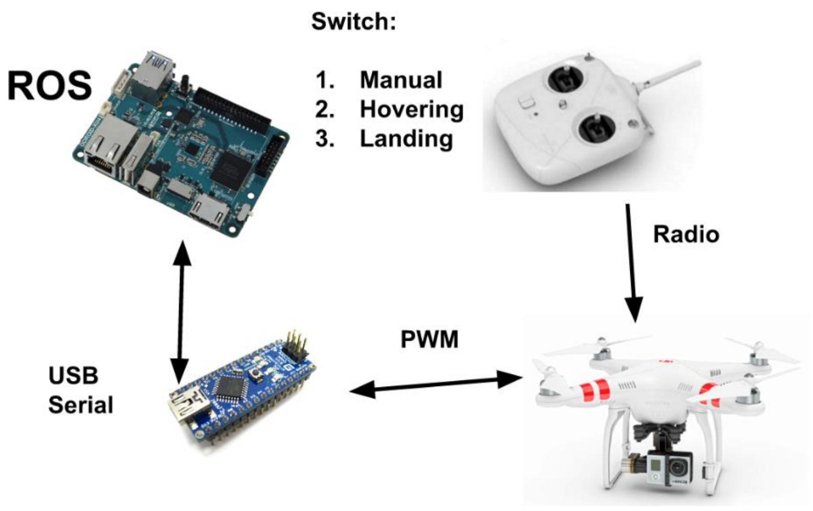

Figure 10: Flight Control subsystem depiction

7.1.4. Power Distribution Subsystem

The power distribution subsystem is responsible for the distribution of power from the standard rotorcraft battery to all onboard subsystems. We created a printed circuit board for this task, but have experienced issues with noise propagation. We then purchased a 3DR power module to regulate the rotorcraft battery down to 5V for the electronics, however this requires messy wiring to ensure all components get powered. Our solution was to mount the 3DR power module to the printed circuit board we created. This printed board allowed us to route the power to all of our devices using standard barrel plugs, while the 3DR module handled the voltage conversion.

7.1.5. Rotorcraft Modifications

The rotorcraft is a DJI Phantom II. This is an off-the-shelf product that is designed as a consumer product, not a development platform. In order to turn it into an autonomous aircraft, several modifications were made. This included landing gear and payload modifications, power hacking, and signal hacking.

We designed and 3D printed new legs for the quadrotor. These legs acted as the landing gear, as well as housed the payload for the quad, which included the sensors, computer, and all other electronics. The image below shows the custom legs with all equipment attached.

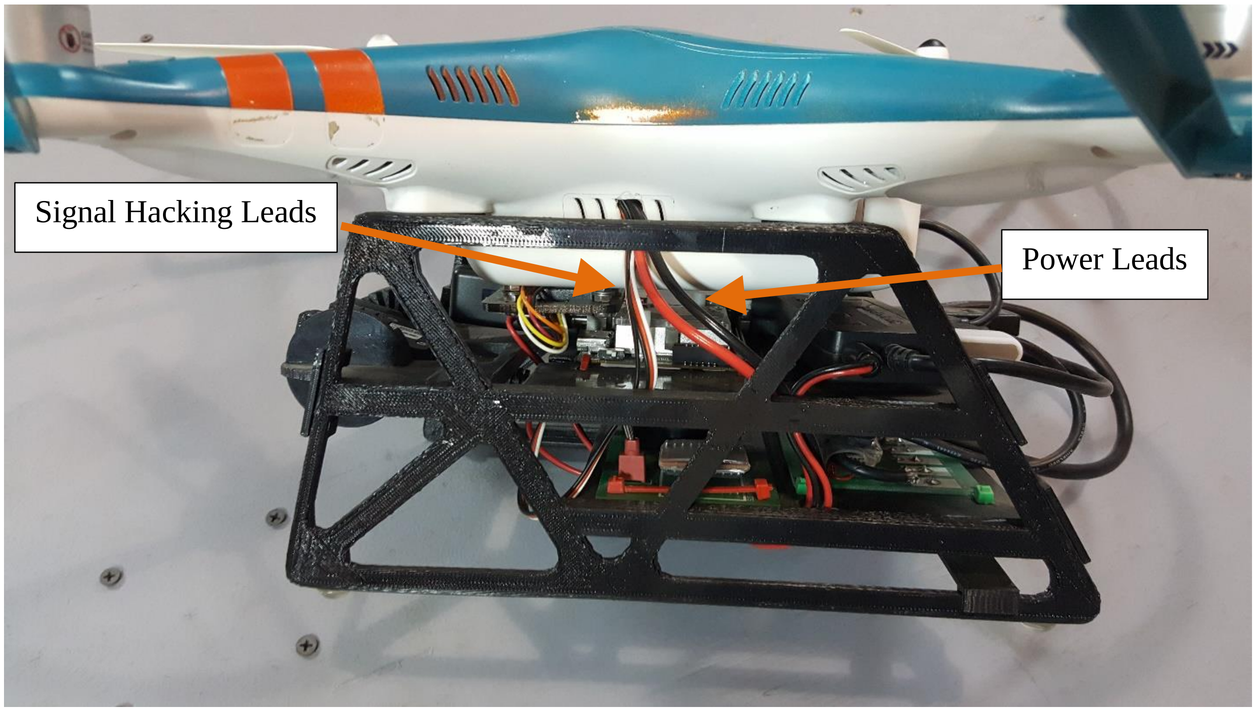

Figure 11: Custom legs with payload

The legs included removable shelves to house all of the components. The removable nature of the shelves were very useful, as it made it easy to replace components and modify the layout as development occurred.

The components needed to be powered, so we soldered wires to the power leads inside the quad. These power leads gave us 11V unregulated power directly from the battery that powers the quadrotor. In the image above, you can actually see the red and black wire coming out of the quad.

In order to control the rotorcraft, we had to intercept the signal that goes from the remote control receiver to the on-board flight controller. The leads we used to intercept this signal are seen in the image above. These were able to connect to an Arduino Nano and control the quad with our Odroid.

7.2. Modeling, Analysis, and Testing

7.2.1. Fisheye Lens

The decision to switch to a fisheye lens for the landing camera was triggered by the loss of the beacons from the field of view of the initial short range camera when the motion from the quad would exceed the angular restrictions imposed by the field of view. In order to combat this so that there would be less losses of the beacons, we decided to switch the short range camera lens to a 180 degree fisheye.

Upon making this change though, we quickly noticed that the distortion caused by a fisheye with this large of a field of view would cause the resulting images to be unusable for computing the pose of the deck. This called for the input fisheye image to be calibrated and rectified. Unfortunately, the camera calibration pipeline currently supported in ROS through OpenCV doesn't yet support fisheye lenses. Ultimately, the solution was to use an omnidirectional camera calibration script in Matlab, then perform the corresponding rectification in ROS using C++ code and basic OpenCV functionality rather than the calibration methods that only hold for the pinhole model such as cv::undistort().

After calibrating out the distortion, the final field of view of the fisheye camera is closer to 130 degrees, however, this may be later improved with a better calibration if we decide we need additional field of view. Compared to the 23 degrees or so that we had before mounting the fisheye, the calibrated image is certainly an improvement and solved many of the problems we were experiencing with the standard plano-convex lenses.

Figure 1 shows an example image both before and after rectification using the new fisheye lens.

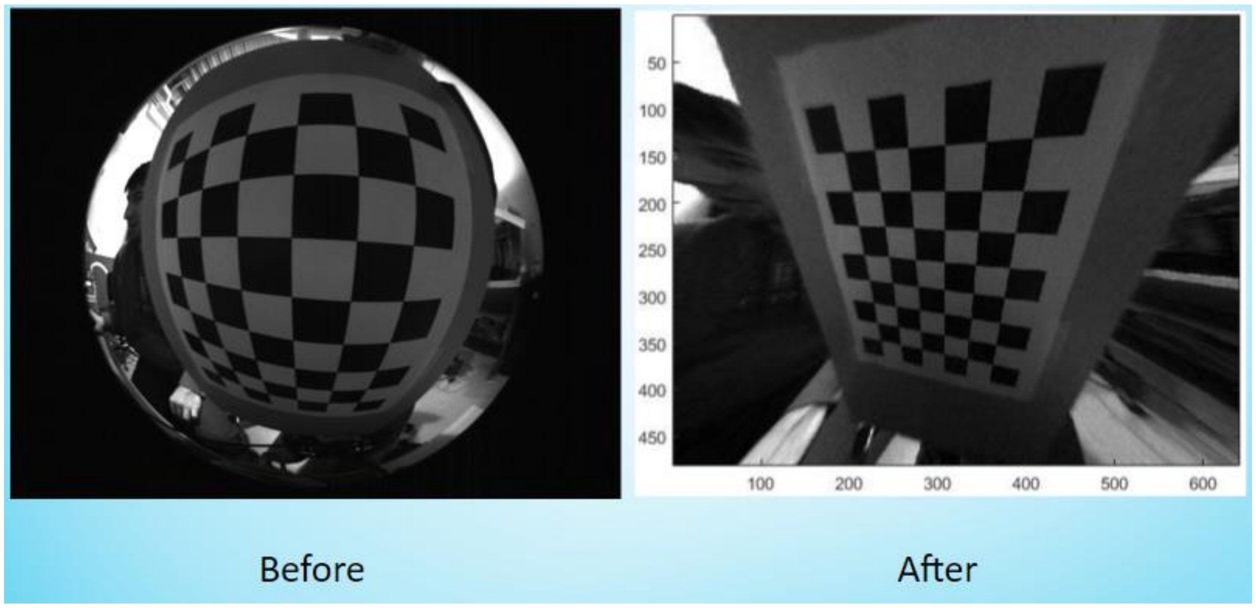

Figure 12: Before (left) and after (right) fisheye camera rectification from calibration

7.2.2. Fisheye Camera Accuracy Verification:

After successful rectification of the fisheye camera lens for the short range landing camera, the accuracy of the localization software needed to be reevaluated. Without reevaluation, we'd have no way of knowing if the assumptions made in OpenCV for the pinhole camera model, assumptions that aren't valid for the raw fisheye image, hold for the rectified fisheye image. In order to test this, I first clamped the camera to a mount on the lab bench. Then, I measured the distance between the fisheye camera lens surface and the center of the deck with a measuring tape. Finally I calculated the distance from the camera to the deck by simple trigonometry using the localization output. Repeating this experiment in the worst case, with the deck beacons near the outside of the rectified image and the distortion in the raw image is greatest, I've determined that the upper bound on our error in localizing the deck is about 4%. As shown in Figure 1, when the deck beacons are closer to the center of the rectified field of view, the error is much less.

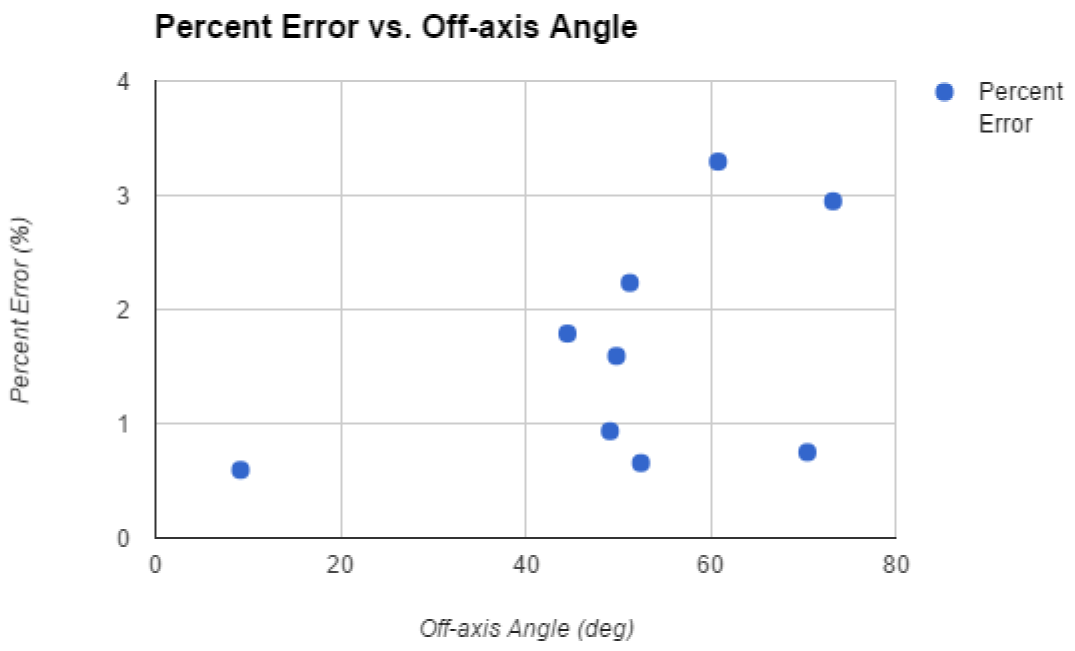

Figure 13: Fisheye localization error quantization

7.2.3. Power Distribution System

Following initial testing, the new power distribution board appeared to work as designed. It steps down the 11.1V input from the quadcopter battery to the 5V needed by the Odroid. When observing the output with an oscilloscope however, we noticed that there was some small noisy disturbances on the ground line.

Upon attempting to power the Odroid, once the boot sequence draws more than an amp, the noise in the ground plane becomes amplified significantly. This causes a brownout in the Odroid that prevents the board from booting. Even first booting the board from its standard power supply then hot swapping to the power board only worked for a few seconds.

The root cause of the noise was never determined, so in the interest of time, we decided to use a 3DR power module to perform the same function, albeit in a less clean manner. The extra routing and hot swap feature from our board are still used with the 3DR module providing the regulation component.

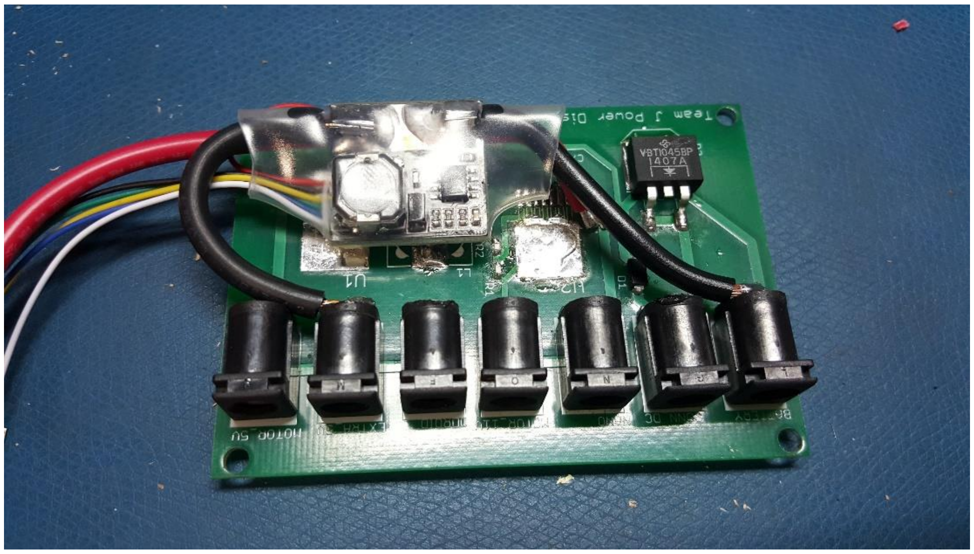

Figure 14: Final power distribution system

7.3. Performance evaluation against the Spring Validation Experiment (SVE)

In this section, we evaluated our system performance by checking the functional requirement, non-functional requirement and our test plan for SVE.

7.3.1. Evaluate with Functional and Non-Functional Requirements

Functional Requirements	**Performance Requirement**	**Result**
Identify the deck within the environment	1 false positive deck identification allowable in 100 trials	Success with 100 % identification rate
	Rotorcraft shall sense position of deck relative to the rotorcraft: 1. Detection at 5m distance 2. 10cm + 4cm/m allowable error	Success
Predict dynamics of the deck	Predict the deck within 4cm/m of current distance to the deck	Success
Rotorcraft shall robustly follow a trajectory	Follow planned trajectory 99% of time, within 0.8cm/m of current distance to deck	Failed with error larger than the requirement
Rotorcraft shall land on the landing zone of the deck.	Rotorcraft shall land within 50cm of center of the deck, as measured from the center of the rotorcraft	Success
	Rotorcraft shall land on a dynamically moving deck	Success
	Rotorcraft shall perform 8 successful landings over a 10 cycle lifetime.	Failed with the landing rate around 60%

Nonfunctional Requirements	**Constraint/Requirement**	**Result**
System shall operate in EMCON conditions.	Operation of autonomous landing mode shall be possible without radio commands and in GPS-denied conditions.	Success
System shall operate with minimal user input.	Autonomous landing sequence shall execute after a single user "go" command.	Success
Rotorcraft shall land safely.	System shall operate without damage to rotorcraft or deck.	Success

For the first functional requirement, our localization subsystem can identify the deck every time and the measurement error is less than 5cm. For the second one, our prediction subsystem can predict the motion of the deck within required criteria. However, for the third one our trajectory subsystem cannot follow the trajectory within 0.8cm/m criteria, so we failed in this test. For the landing test, our quadrotor can land on both static and dynamically moving deck within 50cm landing range. However, the successful rate is around 60%.

Our system satisfies all of the non-functional requirements. It can operate in GPS-denied conditions, the user interface in our system is very simple, and our system operates without any damage.

7.3.2. Evaluate with SVE test plan

Test Plan	**Criteria**	**Result**
Stable Hovering	Quad shall not display jerky movements	Success
	Quad does not move greater than 30cm away from original position during hovering	Success with the error less than 30cm
Landing in a static deck	Quad lands within three minutes of switching to autonomous mode	Success with landing time less than 30 seconds
	Quad lands without damage to itself, the deck, or any surroundings	Success
	Quad lands within 50cm of the center of the deck	Success
Landing on a dynamically moving deck	Quad lands within three minutes of switching to autonomous mode	Success with landing time less than 30 seconds
	Quad lands without damage to itself, the deck, or any surroundings	Success
	Quad lands within 50cm of the center of the deck	Success

For the first test, stable hovering, our quadrotor is able to do stable hovering within the error less than 30cm. For both the landing test, our quadrotor is able to achieve the requirements.

According to the result mentioned above, we completed most of the requirements in SVE, and still need to improve the robustness of our trajectory following subsystem.

7.4. Strong Points

One of the strong features our system is how capable it is given its limited sensor package. We’re able to achieve our goals without the aid of GPS, optical flow sensors, laser rangefinders, LiDAR, ultrasonics, or radars. Ultimately, our system merely requires a deck with some IR beacons, the IMU required for the quadrotor platform to fly anyways, and a single monocular camera with a wide-angle fisheye lens.

Another advanced feature is the localization system's invariance to lighting. Our system can function in a broad spectrum of lighting conditions, ranging from total darkness to full illumination. Though the system was never tested in direct sunlight, its ability to detect and filter out sources of noise such as flashlights, LED's and reflections, we are confident in its ability to perform under all reasonable lighting conditions.

Fisheye lenses are seldom used in precision applications due to the large distortion in the lens. Our system utilizes a combination of camera calibration techniques in order to rectify the fisheye camera image to be useful for precision operations such as solvePnP. Out calibration procedure preserves more than twice as much angular field of view versus the stock ROS camera calibration tool.

Or trajectory generation algorithm gives a quintic polynomial trajectory for velocity and acceleration as well as position. This allows for the use of the acceleration term in our flight control as a form of feed-forward control. This capability was critical to our being able to achieve enough control of the quad.

7.5. Weak Points

One of the weaker parts of our system is our repeatability. At the height of our performance capabilities, we only achieved a success rate for static landing of about 3 in 4 attempts and for dynamic the ratio dropped to around 1 in 2 attempts. This was likely a result of varying atmospheric conditions such as air flow and pressure, something we determined our quad to be very sensitive to. This was likely due to our inability to intercept and regulate the barometer input signal to the DJI internal flight controller.

Another weak point for our system is the tradeoff made for the wide angle fisheye lens. The wider angle is still distributed across the same sensor area, resulting in less effective resolution. This limits the range at which the localization algorithm is effect to about 5m, which is relatively short.

Some assumptions are made with respect to the prediction algorithm, such as the periodicity of the deck. Furthermore, it is assumed that the motion of the deck is relatively simplistic.

8. Project Management

8.1. Schedule

Our original scheduling strategy was to create milestones at two or three week intervals. These intervals were largely structured around presentations, progress review, demonstrations, or other class scheduled deadlines. These milestones included minor functionality improvements or additions such as “Get Cameras Working in ROS” or “3D Print Legs”.

In the first semester, these broad milestones were enough to keep the project moving forward when coupled with the hard deadlines imposed by the class structure. As the semester progressed and in the second semester, this scheduling structure began to fall apart. The milestones were too general and too spaced out. This resulted in a lack of accountability and tasks either slipping or team members being underutilized.

Additionally, our initial milestone approach wasn’t structured correctly. The original structure pushed all subsystems separately and then left some time before the end of each semester for integration. Ultimately, integration was one of the toughest and time consuming steps in the project.

After the beginning of the second semester, the team realized that the current scheduling system wasn’t working. We switched gears to an Agile-like system. Each team member would focus on a specific task or “sprint” for a few days or even up to a week. Then at the end of each week or over the weekend, the team would work together to integrate the new changes and bring the performance of the while system up to par before splitting to work on individual tasks again.

This system allowed for delays or blocking actions to be recognized more swiftly and addressed by the entire group. The tighter, more specific schedule allowed for more collaboration when needed, and clearer objectives for each individual team member to accomplish.

Moreover, this new system allowed for the functionality of the system in its entirety to make gradual improvements, rather than having individual subsystems almost complete which others lagged behind.

8.2. Budget

Table 2 below shows a summary of the budget for the project. We spent almost the entirety of the MRSD budget. Our sponsor helped us by providing one of our quadrotors, a set of infrared beacons, and a mechanical actuator for our deck.

Table 1: Budget summary for project.

	MRSD Budget	**Sponsor Budget***	**TOTAL BUDGET**
Initial	**$4,000.00**	**$5,000.00**	**$9,000.00**
Quadrotor (with spares)	($822.68)	($800.00)	($1,622.68)
Optics	($1,807.08)		($1,807.08)
Computation	($352.98)		($352.98)
Electrical	($22.41)	($100.00)	($122.41)
Mechanical	($361.00)	($200.00)	($561.00)
Other	($307.57)		($307.57)
Spent	**($3,673.72)**	**($1,100.00)**	**($4,773.72)**
Remaining	**$326.28**	**$3,900.00**	**$4,226.28**

* Sponsor budget values are approximate

8.3. Risk Management

Risk #	Risk	Type	Description	Likelihood	Consequence	Likelihood Reduction Plan	Consequence Reduction Plan
1	Unstable Autonomous Flight	Technical, Schedule, Budget	Autonomous landing mode does not control flight adequately/correctly, causing DJI to maneuver in unsafe ways.	4	2	RC switch for manual override	Order extras parts
2	Light interference in vision system	Technical	Vision system identifies non-beacon source of light as beacon, causing localization to give inaccurate positions causing erratic flight behavior.	5	4	Infrared pass filter, testing indoors	Beacon tracking subsystem implementation
3	Asynchronous timing	Technical	Algorithms do not correctly interpret timing of signals	2	5	Create publish rate dependencies on subscribing nodes	Early implementation for troubleshooting
4	Test location unavailable	Schedule	Testing is unable to be done at the planned location due to distance, access limitations, timing, or weather conditions.	2	3	Early coordination with sponsor	Construction of small test platform to be modular with large test platform
5	Payload too heavy	Technical	Weight of sensors and circuitry is too heavy for DJI to perform flight maneuvers effectively	1	5	Weight tracking of components and payload testing	Identify and limit unnecessary weight
6	Printed circuit board nonfunctional	Schedule, technical	Printed circuit board not designed correctly, resulting in rework, unpowered parts, or damaged components	4	4	Perform all component planning prior to ordering board	Define and order needed parts early
						Get board reviewed by TA or professor	
7	Microcontroller dies during flight	Technical	Microcontroller for forwarding flight commands loses functionality during flight, causing DJI to crash	2	5	Robust testing in controlled environment	Order spare parts
8	Incompatibility between single board computer and ROS architecture	Technical	Software drivers for hardware components are not compatible with single board computer	5	4	Investigate OS requirements of hardware prior to purchasing and development	Reserve budget for acquiring alternative sensor or SBC

1. Unstable Autonomous Flight:

During the autonomous flight test, we always have a pilot ready to take over control to prevent any crashes. We also have propeller guards to prevent further damage to the quadrotor and extra propellers in case of crashes.

2. Light interference in vision system:

The performance of the vision subsystem used to be prone to interference from light sources in the environment. We implemented a beacon tracking subsystem to distinguish the actual beacons on the ship deck and the environmental light sources. The result was successful.

3. Asynchronous timing:

To prevent asynchronous timing issue, we implement a state machine. This state machine subscribes to every subsystem and reroutes all of the necessary topics and signals. In addition, the state machine has an error indicator connected to a blinking LED to alert the pilot of the status of the machine.

4. Test location unavailable:

Throughout the development, we built a small platform for indoor testing and finally used it for our final demo. The result was successful and meets our requirements.

5. Payload too heavy:

Several weight reduction plans were implemented to reduce the payload of the quadrotor. We built a new printed circuit board for the flight control subsystem. The 3D printed landing gear was redesigned to have less weight and a stronger structure. We ordered shorter USB cables for cleaner routing and less weight.

6. Printed circuit board nonfunctional:

We revised several versions of our printed circuit boards. The printed power board was not functional under standard operating conditions. The risk mitigation plan was to combine our design with a commercial product. The result was successful.

7. Microcontroller dies during flight:

The microcontroller died once during development. The reason for failure was due to the disconnection of the USB cable. Our solution was to enhance the strength of the connector with wires to prevent it happening again.

8. Incompatibility between single board computer and ROS architecture:

The incompatibility issue was solved by using different drivers and creating a separate version control repository.

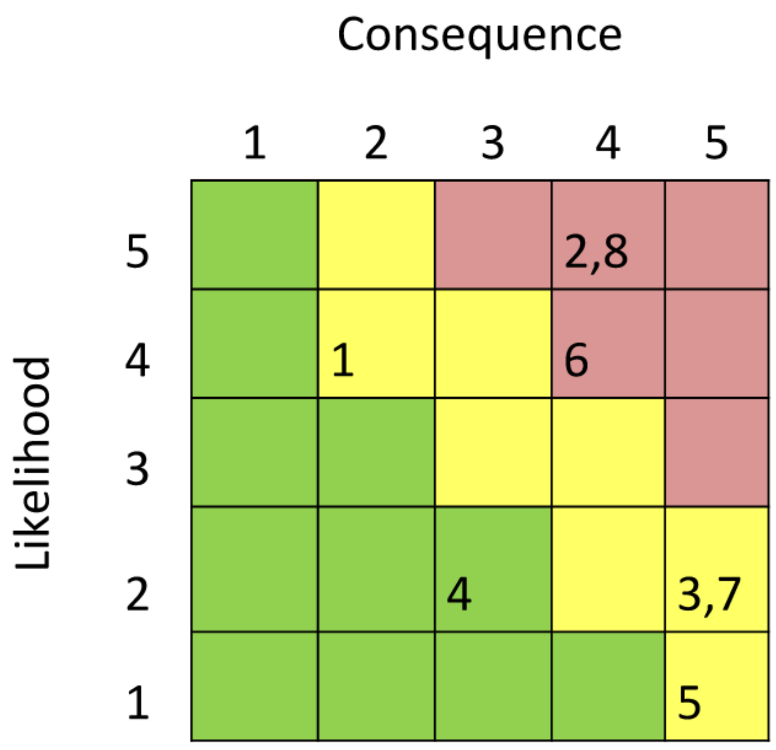

Figure 15: Consequence/Likelihood risk chart

9. Conclusions

9.1. Key Fall semester lessons learned

The first lesson we learned from this project is the underestimation of the workload. With our limited experiences, we were not able to accurately estimate the task length and workload. Uneven workload distribution results in blocking actions and delay schedules.

After the project definition with our sponsor and MRSD advisors, we only have one month left to finish our task. However, we promised to finish a prototype that can land on a static deck in Fall semester. We should have a more realistic goal. Instead of promising to have all subsystems integrated together, we should define our requirements on each individual subsystem first.

Most of our team members don't have enough knowledge in software development. We spent a great amount of time learning ROS, github, and single board computer. In the perspective of learning, we did a great job. In the perspective of the project development, this results in delay of the development and a great burden on the software person in our team. In addition, we should have start our version control before any software development, especially syncing all the results with our single board computer. Due to the delay arrival of our single board computer, the compatibility issues between our laptops and the single board computer already pile up by the time we integrate our codes into it.

A majority of our work has already been done in this field. Our sensors and algorithms are all well developed. Yet we didn't ask for help often enough. We have a great sponsor that can provide us technical support but we only report our progress during weekly meeting. By the time we realized we need help, it was already too late to make any changes due to the deadline of FVE.

We made our schedule too tight to account for any unexpected problems. There are many tasks that is hard to foresee. Our management needs to be more agile to meet our goals. For example, we promised to demonstrate the quadrotor flying using the integration results of the trajectory generation and flight control subsystems at the 4th progress review. To meet this goal, we not only need the software to work with each other, but also need the single board computer, the power distribution system, and the sensor mount to be ready to use. However, we didn't plan these tasks in our schedule. Our solution to this problem was to plan a mini schedule for each member to work on these unscheduled tasks. The results were pretty successful. We need to employ this method more frequently.

Although we have team meetings very frequently and we all know the progress of each subsystem. But when one subsystem goes wrong, we still need that person to fix it. We initially assigned two people responsible to one subsystem but we didn't implement it. This results in unnecessary delay to our schedule. This problem should be avoided in the spring semester.

For the spring semester, the biggest lesson we learned is system integration. Although we all know how hard it is and try to minimize the mistakes that could have been made, we still spent a lot of time on it. The first lesson is the topic naming convention problem. We should have a more clear naming convention during development. The second lesson is that we should use the error state indicator earlier. Often times we don't know the root cause of one failure and we did not log the data every time. The error state indicator gives us more information about the system and increases the efficiency of the overall development.

9.2. Future Work

There are several aspects of this project that could be improved on future iterations. The primary areas we think would be good places to improve are in control and prediction. There are also a few known bugs that could easily be fixed with additional time and effort.

9.2.1. Control Improvements

The controller could be improved by determining the system dynamics more precisely. The current algorithms assume that the controller stick input is linearly mapped with the motion of the quad, which is not true. Determining this actual mapping would help to improve the controller and allow for more precise and repeatable motion.

9.2.2. Prediction

The current prediction algorithm only predicts motion in one degree of freedom – the roll of the deck. It assumes that the motion is periodic, and does not work if the period of the motion is not consistent for a period of time. This could be improved to include several degrees of freedom, as well as more exotic deck motion. Ideally, the deck could be moving in six degrees of freedom in an arbitrary way and the prediction algorithm would classify the motion and predict it forward to evaluate the safe landing times.

9.2.3. Known Bugs to Fix

There were a few known bugs that were not prioritized to complete this project. First, when the beacons are at the edge of the field of view of the camera such that they are going in and out of view, the prediction algorithm decides that the deck is dynamically moving, even if it is not. This could be fixed with a few more logical checks.

Next, once the quad sees that the deck is dynamically moving, it takes a long time looking at a static deck to decide that the deck is actually static. This proved to be a bigger issue when combined with the bug mentioned above. If the quad thinks the deck is dynamic and we try to land, it will hover above the deck indefinitely, rather than going in for a landing. This could also easily be fixed by upgrading the logic for determining if the deck is static or dynamic.

10. References

[1] http://www.webdesignschoolsguide.com/library/10-things-we-couldnt-do-without- robots.html

[2] https://upload.wikimedia.org/wikipedia/commons/7/74/US_Navy_091031-N-1251W-009_An_SH-60B_Seahawk_helicopter_assigned_to_Helicopter_Anti-Submarine_Squadron_Light_(HSL)_51_approaches_the_guided-missile_destroyer_USS_Lassen_(DDG_82)_during_deck_landing_qualifications.jpg