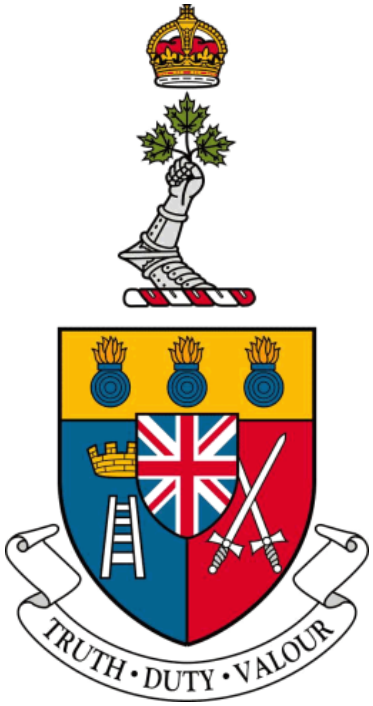
Royal Military College of Canada

Department of Electrical and Computer Engineering




# Designing Coatimunde

Computer Optics Analyzing Trajectories In Mostly Unknown, Navigation Denied, Environments

DID-07 - Detailed Design Document

**Presented by:**

Amos Navarre Hebb & Kara Stephan

**Presented to:**

Dr. Sidney Givigi?? & Dr. Anthony Marasco & Dr. ?????? ??????

March 10, 2019

# Contents

**1 Introduction** **4**
1.1 Document Purpose . . . 4
1.2 Background . . . 4
1.2.1 Aim . . . 4
1.2.2 Benefits . . . 4
1.2.3 Scope . . . 5
1.2.4 Requirements . . . 5
1.2.5 Preliminary Design - NOT DONE . . . 5
1.2.6 Changes to Schedule - NOT DONE . . . 5
1.3 Definitions . . . 5
1.3.1 Obstacle Avoidance . . . 5
1.3.2 Unmanned Aircraft Systems . . . 5
1.3.3 Computer Vision . . . 6
1.3.4 OpenCV . . . 6
1.3.5 Gazebo . . . 6
1.3.6 Robot Operating System . . . 6
1.3.7 TurtleBot . . . 6
1.3.8 AscTec Pelican . . . 6

**2 Design** **7**
2.1 Computer Vision . . . 7
2.1.1 Target Finding . . . 7
2.1.2 Obstacle Finding . . . 7
2.1.3 Optical Flow . . . 7
2.2 Analyzing Trajectories . . . 7
2.3 Unknown and Navigation Denied Environments . . . 7
2.4 Block Diagram . . . 8
2.5 Custom Nodes . . . 9
2.5.1 Target Finding . . . 10
2.5.2 Obstacle Finding - NOT DONE . . . 10
2.5.3 State Estimator . . . 10
2.5.4 Route Planning - NOT DONE . . . 11
2.6 Node Diagram . . . 12
2.7 Mathematical Modeling - NOT TOUCHED . . . 12
2.7.1 Flying Robot . . . 12
2.7.2 Robot in 3D Space . . . 12
2.7.3 Pose Estimation . . . 13
2.8 Interfacing . . . 13

**3 Equipment** **13**
3.1 Equipment Table . . . 13

**4 Verification and Validation** **13**
4.1 Unit Tests . . . 13
4.2 Gazebo . . . 13
4.3 Camera Tests . . . 15
4.4 Ground Based Robot . . . 15
4.5 Flying Robot - NOT DONE . . . 15

**5 Results - NOT DONE** **15**

**6 Discussion - NOT DONE** **15**
6.1 Functional and Performance Requirements . . . . . 15
6.2 Interface Requirements . . . . . 17
6.3 Simulation Requirements . . . . . 17
6.4 Implementation Requirements . . . . . 17
6.5 Schedule Requirements . . . . . 17

**7 Conclusion** **17**

# 1 Introduction

## 1.1 Document Purpose

Using Computer Optics for Analyzing Trajectories in Mostly Unknown, Navigation Denied, Environments (COATIMUNDE) is the goal of this project. The purpose of this document is to outline the detailed design for COATIMUNDE. That is, any deviations from the original design/plan, what the final design was, how it was built, and how this met the requirements of the project. The requirements for the project have been outlined in the Statement of Requirements. This design of the project is shown, through design artefacts, and discussed in the Design Section. The final results of the project are presented in the Results section, and the tests done to obtain these results are presented in the Verification and Validation section. This document will then provide a summary of the degree of success of the project and provide feedback on the course experience as a whole.

## 1.2 Background

Both in the consumer and professional sectors the use of autonomous aerial vehicles is growing quickly. Currently these vehicles rely on skilled pilots to accomplish a very limited set of tasks. Adding obstacle avoidance capabilities to these vehicles and simplifying the task of following targets could allow for these systems to be used in many more situations. This section will give a quick background on obstacle avoidance, unmanned aircraft systems, computer vision, and the platforms we intend to use in this project.

### 1.2.1 Aim

The aim of this project was to design a high level control system that will allow an air robot to identify a target and move toward it, avoiding any obstacles that are in the way. Tracking targets of interest in complex environments with a flying robot was the ultimate goal of this project. To accomplish this goal we used a TurtleBot and then a flying robot through only the use of a camera to identify targets and obstacles.

### 1.2.2 Benefits

Having a flying robot capable of accomplishing the project's tasks while totally autonomous will allow for the use of flying robots in environments closer to the ground, and will assist pilots in complex environments. These general requirements can be used in many situations. These benefits for society are the motivation behind this project.

- **Surveillance:** The robot could follow an interesting object, especially in an urban environment, without colliding with obstacles.
- **Search and Rescue:** The robot could move toward visual way-points set by pilots while avoiding obstacles in complex environments assisting in search efforts.
- **Inspections:** The robot could inspect objects in hard to reach environments and complex environments like rooftops or bridges.
- **Disaster Relief:** The robot could check inside buildings that may have compromised structural integrity, rubble on ground, *&c.*
- **Agriculture:** The robot could inspect tree-fruit or crops that can not be observed from overhead or check assets in remote locations such as irrigation equipment.

#### 1.2.3 Scope

The project started through the use of a ground robot then progressing to a flying robot. The project used computer vision to find targets and for avoiding multiple obstacles. Both robots are be highly autonomous, requiring only user input to commence.

The scope of the project was limited due to only being able to test indoors. Ideally the UAV would be operating at a high speed and identifying an arbitrary target in an unpredictable environment. In this case though the testing was completed within the confines of a relatively small robotics laboratory. Smaller obstacles were used indoors simply due to space constraints. Lower speeds were used as well, again due to the lack of space.

#### 1.2.4 Requirements

The requirements for the project are described in the Statement of Requirements (DID-03) [1]. The main functional requirements for this project were identification and movement towards a target within 15m radius, and identification and avoidance of multiple obstacles.

The initial requirements show certain assumptions that were made which can be attributed to ignorance at the time. The most notable of these, the use of a single video stream and exclusively OpenCV to parse depth data. It proved very difficult to extract reliable depth information from a single camera. Although we initially had identified a method using stereo cameras, the push-broom approach [2] we had initially explored, we had abandon this due to a belief that depth from a single camera would be similarly visible, and constrained ourselves in the requirements phase to this.

#### 1.2.5 Preliminary Design - NOT DONE

The preliminary design document outlines out the initial design for the project [3]. The initial design was fairly robust, but certain aspects had intentionally not yet been selected. The main block diagram Figure 1 remains totally unchanged, the only additions are the specific topics being subscribed to by each of the custom modules.

#### 1.2.6 Changes to Schedule - NOT DONE

- delay of getting the robot to identify targets and move towards them (REF schedule update) [4] - change of the layout of custom nodes (can send them to design section for custom nodes?) - delay of porting to UAV

### 1.3 Definitions

#### 1.3.1 Obstacle Avoidance

Obstacle avoidance is the task of satisfying a control objective, in this case moving toward a visual target, while subject to non-intersection or non-collision position constraints. The latter constraints are, in this case, to be dynamically created while moving in a reactive manner, instead of being pre-computed.

#### 1.3.2 Unmanned Aircraft Systems

Very generally any powered vehicle that uses aerodynamic forces to provide lift, without a human operator being carried, can be considered an unmanned aerial vehicle. Currently most of these vehicles make up a single component of a larger unmanned aircraft system. An Unmanned aircraft system (UAS), or remotely piloted aircraft system (RPAS), is an aircraft without a human pilot on-board, instead controlled from an operator on the ground. Such a system can have varying levels of autonomy, something as simple as a model aircraft could be considered a UAS without any automation capabilities.

### 1.3.3 Computer Vision

Currently there are many different ways that computers can make high-level decisions based on digital image information. There are many methods to acquire, process, and analyze data from the real world using a camera. While this is a very broad field, we intend to focus on motion estimation and object recognition. Both will be working with a video stream taken from a camera.

Motion estimation can be accomplished using direct methods which compare whole fields of pixels to each other over successive frames, compared to indirect methods which look for specific features. The information resulting from motion estimation streams can be used to both compensate for motion while analyzing other aspects of an image, and update a state machine.

Object recognition in our project will be accomplishing two tasks: identifying a marker or target which will require more involved object recognition calculations, and very simple techniques, such as edge detection, to identify obstacles that exist in the path of the robot.

### 1.3.4 OpenCV

The Open Source Computer Vision Library (OpenCV) of programming functions is a cross-platform and free for use collection of functions primarily aimed at real-time computer vision[5]. Most well documented techniques to accomplish all of the computer vision goals of our project have already been created and refined in OpenCV. For this reason this project utilized a lot of preexisting OpenCV functions.

### 1.3.5 Gazebo

Gazebo is a robot simulator that allows for creation of a realistic environment which includes both obstacles and markers similar to those being used in the lab. It was then used to rapidly test algorithms.

### 1.3.6 Robot Operating System

The Robot Operating System (ROS) is a distributed message system that allows for various sensors and processors to work together to control a robot. It is open source and has been integrated already with OpenCV and Gazebo. There are many additional tools for detecting obstacles, mapping the environment, planning paths, and much more. It is also a robust messaging system that has been proven to be capable of real-time processes.

### 1.3.7 TurtleBot

The TurtleBot is a robot kit with open-source design and software. A TurtleBot is a robot built to the specification for TurtleBot Compatible Platforms[6]. In this case the turtlebto has a Kobuki Base, an Acer Netbook running Ubuntu with ROS packages added, an X-Box Kinect, and some mounting plates.

The resulting robot looks like a disk supporting some circular shelves with a small laptop and a camera on the shelves. The base disk is 35.4cm in diameter, the topmost shelf is 42cm from the ground. The robot has a maximum speed of 0.65m/s.

### 1.3.8 AscTec Pelican

The Ascending Technologies Pelican is a 65.1cm by 65.1cm quad-copter designed for research purposes[7]. It includes a camera, a high level and low level processor set up for computer vision, and simultaneous localization and mapping (SLAM) research. It is also capable of interfacing easily with other controllers and can carry up to a kilogram of additional gear.

# 2 Design

## 2.1 Computer Vision

Computer optics was used as the robot employed a camera to identify objects in the surrounding environment, specifically targets and obstacles. There are three unique manners that computer vision was used as an input source for this project; these are target finding, obstacle finding, and optical flow. The implementation of all three created a navigation kit that uses exclusively a camera as input resulting in a very transferable and lightweight package for flying robots.

### 2.1.1 Target Finding

While finding arbitrary targets would be preferred, this project is going to be limited to finding special targets designed to be easily identified in a busy environment called ArUco shapes. There are existing libraries in OpenCV that are useful for identifying ArUco shapes with very little overhead.

### 2.1.2 Obstacle Finding

Finding obstacles will be a considerable aspect of this project. There are various algorithms which leverage the robot's motion through the environment to extract key features of the environment, indicating the potential presence of obstacles. Parallax shift is the tendency for items that are closer to a camera to appear to move more or change in size more than a background that is further away and can be used to indicate where items are and how close they are. Occlusion is where one item moves in front of another item and covers it up, this indicates that the item still in view is closer to the camera than the item that has been hidden. OpenCV contains libraries for both of these tasks, as well as others that may end up being employed to identify obstacles in the environment.

### 2.1.3 Optical Flow

Optical Flow is a possible source of motion information for the robot. As a camera is moved through an environment the entire image will appear to scale larger as the camera moves forward, rotate left and right as a robot rolls, and shift left, right, up, and down as the robot pitches and yaws. Calculating shifts from one frame to another for global shifts caused by movement of the camera is a mathematically intensive process beyond the scope of this project. OpenCV contains libraries to try to parse this information, but it requires a considerable amount of effort to combine this information with a model of a robot to get useful position estimation.

## 2.2 Analyzing Trajectories

Analyzing trajectories is a term that refers to the robot completing movements to go towards the target. To do path planning the project will implement potential fields. Potential fields function through creating vectors pointing to objects in the environment and assigning them positive or negative. The positive vectors attracts the robot towards the target and the negative makes obstacles repulsive to the robot [8]. This algorithm was selected to do path planning as the robot will not have to build a map of the environment to create path to the target [9]. Potential fields allow for the robot to change its path along the way if it discovers new obstacles. As the robot is moving the potential fields must be updated to account for how much the robot moved. This will be solved through creating a state machine.

## 2.3 Unknown and Navigation Denied Environments

The robot must always remember where identified target and obstacles are in relation to itself even when its own position changes. Navigation denied means that the robot will not have access to a GPS to locate itself, so it must search its unknown environment to identify targets and obstacles. An unknown environment can be defined as an environment in which the robot has no prior knowledge of

its surroundings and must identify objects in this environment itself through computer vision. Potential fields will be used to track the location of the target and obstacles through updating each objects vector. These vectors will be used in the state machine to properly create a path to the target, as they are either negative for the target or positive for obstacles. This once again is the reason that the project will be using potential fields to plan the path to the target.

## 2.4 Block Diagram

The block diagram that is shown in Figure 1 contains the outline of the COATIMUNDE project. It is a very high level overview and does not contain references to most of the hardware in use for two reasons: the actual robot and laptop being used are relatively arbitrary as long as the software included is present on both. ROS also includes many nodes which are not represented on the block diagram. Only the nodes that are most important to our project, have been specifically added to ROS, or have been custom created are included on our system block diagram shown below.

The laptop contains the initialization, target selection, and RViz. The laptop also takes the user input when necessary and processes it appropriately. The robot does all the target identification, obstacle avoidance, and movement decisions autonomously. This means it takes the input from the camera into a video processing node and then create a corresponding ROS message to be sent to the other systems on the robot. These messages are created and used throughout the different systems on the robot. The model of the robot's environment is found through subscribing to messages from the obstacle finder node, target finder node, and the state estimate node. This model then creates a message to pass to the route planning node which tells the movement node how to update its position and speed.

Most of the nodes used in this project are either being used stock, or are very minimally modified from provided libraries. The four nodes, highlighted in grey, which have been custom built for this project are the State Estimate, Target Finder, and Obstacle Finder. Along with each of these functional names assigned, the actual nodal name for each of the project's custom nodes are listed.

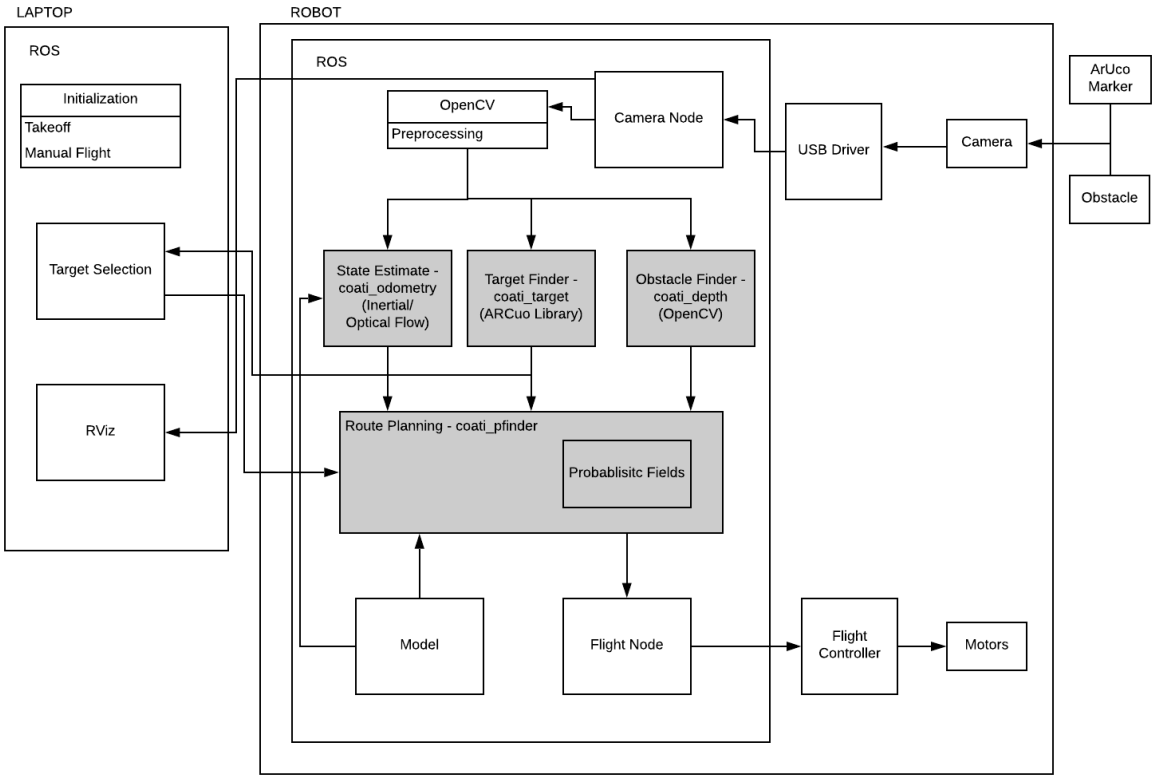



Figure 1: Project Block Diagram

## 2.5 Custom Nodes

All of the nodes that were written for this project were written in Python and then built in ROS to be implemented on the robot. Python was used for this project because C++ and Python are the most supported languages for both ROS Nodes and OpenCV. While C++ could possibly be faster in execution, we believe that most of our bottlenecks occur from the actual processing of images. The library calls to OpenCV were still executing in the lower level C that OpenCV was written in, so there would be minimal gains had we written it in C++.

The Camera node has been referenced in many locations. The camera node is built into ROS and provides image data in various formats. Processes can subscribe to these different image formats. In general the raw image information was used by some of the custom nodes and many other built in ones. Common transformations that must be applied to all photographs will be added to the camera node. These transformations are to correct for the distortion that any camera has due to its own geometry.

### 2.5.1 Target Finding

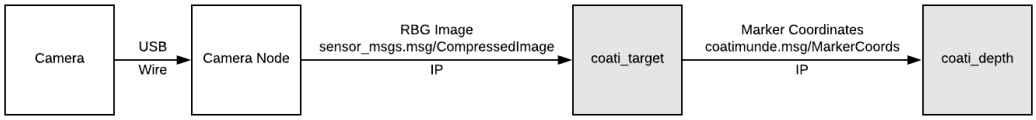



Figure 2: Target Finding Node

Camera data, after being processed by the camera node, goes into the target finder node. The target finder is mostly implementing the ArUco symbol finding libraries in OpenCV. ArUco symbols are intended for augmented reality purposes and provide very accurate position, orientation, and distance information. These values are inserted into a marker coordinate message that the other systems on the robot can understand and be published at least 10 times per second.

Given the limited field of view of the robot, it cannot always be able to see the target. The robot, in the pfinder node, saves a dictionary of points that are associated with targets that have been resolved. Only the most recent 'citing' of a target is recorded, but as the robot moves its position relative to that target is updated as well.

### 2.5.2 Obstacle Finding - NOT DONE

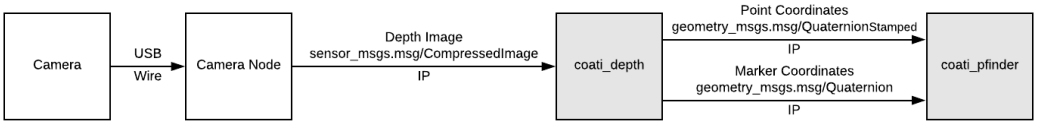



Figure 3: Obstacle Finding Node

Finding obstacles with depth information is difficult using monocular vision. The current plan is to try to combine parallax shift with edge detection to try to parse information about nearby objects. These vectors will be published as a large ROS message at least 10 times per second.

Getting depth information from a single camera is difficult. The TurtleBot also has a suite of depth sensors built in with the camera so these may be used initially instead of trying to use parallax information from a camera to allow for testing other aspects of the system.

The depth node subscribes to the target node to get the target locations. The node then uses the given location to find the depth value for the target. This target depth value is transformed in a quaternion and passed off to the path finder node.

### 2.5.3 State Estimator

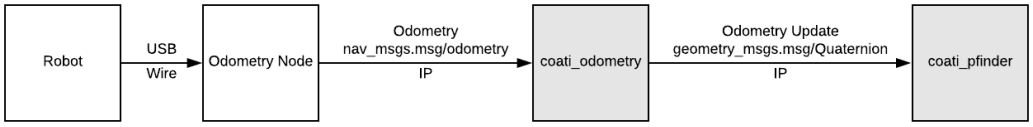



Figure 4: State Estimating Node

State estimating from camera information is very difficult and computationally expensive. The TurtleBot has stepper motors that can measure distances fairly reliably, and the Quadcopter has accelerometers that provide changes in position more reliably than image information could.

These simpler forms of measuring the robot moving through the environment were used and the odometry node subscribed to the robot's odometry updates. This node turns all state updates into quaternions and publishes them for the route planner. The route planning node then adjusts the vectors to nearby obstacles and targets according to how much the robot has moved.

New position estimates should be provided at least 10 times every second in the form of a quaternion from the last position that the robot was in passed over a ROS message.

### 2.5.4 Route Planning - NOT DONE

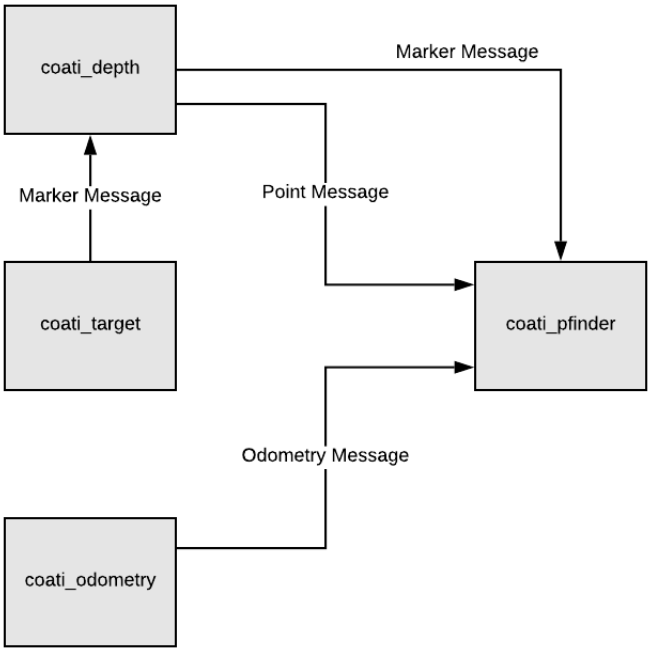



Figure 5: Route Planning Node

Route planning takes input from the other three custom nodes and determines a route that should carry it toward the target. The implemented solution was to create a potential field where positive vectors that repel the robot are attached to all obstacles while a large negative vector is attached to the target location.

As the route node receives more information about the robot moving through the environment it applies transformations to all of the contained vectors to keep them up to date with where they are as the robot changes position in the environment. Target locations are all stored in a separate array to remember the location of every target the robot finds. Differentiating targets is simple as each ArUco marker has an associated identification value. These values allow the node to know when a target has been re-spotted and over-write it or allow for finding and remembering the positions of multiple targets. If it discovers a new target it simply adds it to the array with its positional information.

-talk about how obstacles are path planned around ?

The method to remove unnecessary or repetitive vectors for obstacles is to have these vectors simply expire after a certain amount of time.

The route planning software provides a direction vector to the robot controlling node 10 times every second. There is no need for the route to be the optimal route given that the environment is unknown.

## 2.6 Node Diagram

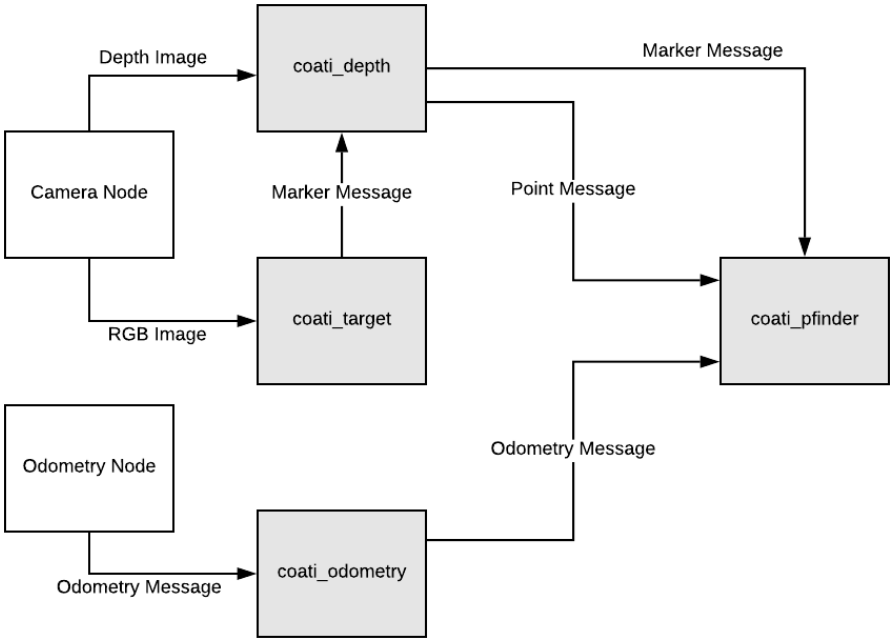



Figure 6: Project Node Diagram

Through Figure 6 it can been shown what nodes were created or used for this project. The diagram displays what node subscribed to the other nodes, such as the target finder node subscribed to the camera node to get messages about the target's potential location. It is important to recognize that the directions indicated in Figure 6 are the flow of messages, not the direction of the subscriptions.

## 2.7 Mathematical Modeling - NOT TOUCHED

### 2.7.1 Flying Robot

Given that a quad-copter has six degrees of freedom, modeling them is a very difficult task. The AscTec Pelican already has many existing models intended for use with ROS. There is also a large collection of hardware agnostic tools for operating a quad-copter.

### 2.7.2 Robot in 3D Space

The robot itself will need to remember where it is located in its environment. Remembering where targets are located relative to the robot is necessary, and presumably remembering the locations of obstacles will be used as well. Using quaternion vectors relative to the robot will allow for simple transformations on these vectors while using very little computational power [10]. ROS contains methods and messages to easily communicate, convert, and apply transformations to quaternions.

#### 2.7.3 Pose Estimation

Initially feedback from inertial sensors will be used for pose estimation, but ideally a version of Optical Flow would be used to estimate the position of the robot in 3D space. There are some libraries in OpenCV that have addressed generalized pose estimation in 6 degrees of freedom, but it requires a significant amount of computational power, a higher frame rate than our goal of 10Hz to be used for real time performance, and hundreds of visual odometry measurements must be made in every frame requiring a visually busy background.

### 2.8 Interfacing

The interfacing of almost all sensors with the on-board computer will be done over USB. Communication between different nodes on the same computer, and any computers off-board the robot, will be done exclusively with ROS messages sent over either internal loops or Wi-Fi.

## 3 Equipment

### 3.1 Equipment Table

!!! Fun fact the equ table is all messy with the custom nodes !!!

A large number of items are being used in this project, most are arbitrary and may be changed in future iterations. Part way through our project we intend to re-implement the entire solution onto a different platform. A naming convention has been developed for all of our components using a three character code.

The first letter, ascending from A for hardware and descending from Z for software, represents the subsystem or general classification the equipment falls under. The second character represents a generic piece of equipment, and the third is for a particular example that we are using for our project. Any part with the same two first letters should be able to be substituted freely, an example may be two cameras. One camera (CAA) may be used on the Turtlebot as it is built in to the Turtlebot frame, while another (CAB) may be used on the flying robot as it is much lighter. The generic camera (CA-) can refer to any camera which provides the same information to the same nodes. The equipment table is shown in Table 1.

## 4 Verification and Validation

–wasn't an actual section before but I think it should be? Or it could go under results for how we obtained the results???

### 4.1 Unit Tests

When using ROS it is typical to create individual nodes that all execute independent of one another. Using this modular design, it is easy to write an entire node and verify it individually before executing it with the entire system. Verification and validation of the project's nodes will be done prior to implementation in the Gazebo simulation environment.

### 4.2 Gazebo

Gazebo is a simulation environment that runs nodes that give similar input and output to nodes being executed on a real robot moving around in a real environment. This includes camera feeds, and allows for the insertion of obstacles and ArUco shapes. Testing of custom written nodes were done in Gazebo congruently with unit-tests to ensure that nodes which rely on subscribing to other nodes are behaving as expected before executing code on an actual robot.

| Purpose | | Description | | Equipment | |
|---|---|---|---|---|---|
| A | Ground Based Robot | A | Robot | A | Clearpath Robotics Turtlebot 2 |
| A | Ground Based Robot | B | Robot Controller | A | Acer Aspire E-11 |
| A | Ground Based Robot | C | Robot Base | A | Kobuki Robot Base |
| A | Ground Based Robot | D | Sensors | A | Stepper Motor Feedback |
| A | Ground Based Robot | D | Sensors | B | Orbbec Astra Pro Sensors |
| A | Ground Based Robot | D | Sensors | C | Gyroscope |
| B | Flying Robot | A | Robot | A | AscTec Pelican Quadcopter |
| B | Flying Robot | B | Robot Controller | A | AscTec Low Level Processor |
| B | Flying Robot | B | Robot Controller | B | AscTec High Level Processor |
| B | Flying Robot | B | Robot Controller | C | ODROID-XU4 |
| B | Flying Robot | C | Robot Base | A | AscTec Pelican Frame |
| C | Camera | A | USB Camera | A | Orbbec Astra Pro Camera |
| C | Camera | A | USB Camera | B | oCam-1MGN-U Plus |
| C | Camera | B | Digital Camera | A | AscTec Option 4 |
| D | Offboard Computer | A | Computer | A | Lenovo T520 |
| D | Offboard Computer | A | Computer | B | Mathworks VM - ROS Gazebo v3 |
| E | Testing Hardware | A | ArUco Symbol | 0-9 | Printed Single Digit ArUco Symbol |
| E | Testing Hardware | B | Physical Obstacle | A | Human |
| E | Testing Hardware | B | Physical Obstacle | B | Box |
| E | Testing Hardware | B | Physical Obstacle | C | Sheet |
| E | Testing Hardware | C | Motion Tracking | A | Reflective Dot |
| E | Testing Hardware | C | Motion Tracking | B | Motion Tracking Cameras |
| E | Testing Hardware | D | Camera Calibration | A | Printed 7cm Checker Pattern |
| F | Interfacing | A | Wireless Router | A | D-Link 2.4 GHz Router |
| S | Linux Operating System | A | Operating System | B | Ubuntu 18.04 Bionic |
| S | Linux Operating System | A | Operating System | K | Kubuntu (Mathworks) 14.04 |
| S | Linux Operating System | A | Operating System | T | Ubuntu 14.04 Trusty |
| S | Linux Operating System | B | Bundled Program | A | Generic Loopback Device |
| S | Linux Operating System | C | Version Control | A | git |
| T | ROS Service - Custom | A | Action Services | A | Set Goal |
| U | ROS Service - Standard | A | Action Services | A | Trigger |
| V | ROS Message - Custom | A | Action Messages | A | Goal |
| V | ROS Message - Custom | A | Action Messages | B | Obstacle |
| W | ROS Message - Standard | A | Sensor Messages | A | Image |
| W | ROS Message - Standard | A | Sensor Messages | B | PointCloud |
| W | ROS Message - Standard | B | Geometry Messages | A | Twist |
| W | ROS Message - Standard | B | Geometry Messages | B | Quaternion |
| W | ROS Message - Standard | B | Geometry Messages | C | Transform |
| X | ROS Node - Custom | A | Depth | A | coati_depth |
| X | ROS Node - Custom | B | Path Finder | A | coati_pfinder |
| X | ROS Node - Custom | B | Odometry | B | coati_odom |
| X | ROS Node - Custom | B | Vision | D | coati_target |
| Y | ROS Node - Standard | A | Vision | A | vision_opencv |
| Y | ROS Node - Standard | A | Vision | B | image_pipeline |
| Y | ROS Node - Standard | B | Coordinates | A | tf |
| Y | ROS Node - Standard | C | Vision | B | image_pipeline |
| Z | Robot Operating System | A | Operating System | A | ROS Indigo Igloo |
| Z | Robot Operating System | A | Operating System | B | ROS Melodic Morenia |
| Z | Robot Operating System | B | Bundled Program | A | Gazebo2 |
| Z | Robot Operating System | B | Bundled Program | B | Gazebo7 |
| Z | Robot Operating System | B | Bundled Program | C | catkin |
| Z | Robot Operating System | B | Bundled Program | D | RViz |
| Z | Robot Operating System | B | Bundled Program | E | rosbag |
| Z | Robot Operating System | B | Bundled Program | F | rqt |
| Z | Robot Operating System | C | Client | A | roscpp |
| Z | Robot Operating System | C | Client | B | rospy |

Table 1: Equipment

### 4.3 Camera Tests

Given that most of the project's custom nodes revolve around transforming camera data to extract information, the ArUco shapes for targets, parallax shift for obstacles, and optical flow for odometry, tests performed with only the camera and OpenCV outside of a node can be created and and executed without being contained within a ROS node. This was useful as one can use OpenCV's feature highlighting capabilities to ensure that the targets and obstacles were being identified while moving a camera through an environment.

### 4.4 Ground Based Robot

Testing the code on a ground based robot was necessary to ensure that the logic in the movement nodes is sound. Initial testing was done through simply having the robot follow moving targets or turn to face targets that are not head on. As obstacle avoidance was getting developed ensuring that the robot does avoid the obstacles was enough. When the project was sufficiently developed the testing was having the robot avoid obstacles and move towards a target. As the project moved closer to porting to the flying robot, the ground robot was tested to ensure it met the requirement outlined in the SOR [1].

### 4.5 Flying Robot - NOT DONE

Similar to the ground based robot, verifying the flying robot primarily involved evaluating its behaviour. The behaviour we expected is the all the functionality that was implemented on the ground based robot previously.

- Once again it is necessary to ensure all requirements are met... - limitations of space, ref scope? Real enviro is only indoors.. I'll grab some issues section for this?

If more complex moves are attempted it may become necessary to test this platform using the motion capture equipment that exists in the lab.

## 5 Results - NOT DONE

- presentation of results - prove we met the requirements

- we need to get some data points to prove the target and obstacles are found - depth readings - able to move towards the target? - talk about what happened with the obstacle avoidance and why it doesn't work

## 6 Discussion - NOT DONE

- detailed analysis of how our design did in relation to the requirements - I made some tables so we just have discuss why we met some requirements and how we didn't met others, it says every requirement must be touched on so I just made a table for each type of requirement... - talk about why some requirements could not be met or only partially met - talk about major issues throughout the project - recommendations if anyone expands on this project in the future – recommendation do something more exciting

### 6.1 Functional and Performance Requirements

The requirements for the Turtlebot can be seen in 2. The reason FR-04 was not satisfied on the Turtlebot is the same reason that was discussed in the results section. FR-05 also was not fully satisfied, the Turtlebot was able to recognize multiple obstacles at once and register each as a potential obstacle. It is the same reason, the obstacle avoidance system, why the requirement was not met. The obstacle avoidance system did not met the requirements on either platform.

| Index | Description of Requirement | Result | Comment |
|---|---|---|---|
| FR-01 | The Turtlebot shall move towards a target under control | Met | |
| FR-02 | The Turtlebot shall recognize optically an OpenCV's ArUco shape and be able to give information on the location of the target relative to the robot | Met | |
| FR-03 | The Turtlebot shall recognize obstacles in its environment and identify where they are relative to itself | Met? | |
| FR-04 | The Turtlebot shall be able to make a deviation from its current movement pattern to avoid an obstacle in its path and then return to its pattern | Not Met | Comment on why or saying we comment below |
| FR-05 | The Turtlebot shall be able to identify multiple obstacles and avoid them accordingly | Partially Met | Able to identify multiple obstacles, not avoid them |
| PR-01 | The Turtlebot shall be able to identify and locate OpenCV's ArUco shapes within a 15m radius | Met | |

Table 2: Summary of Results for the Turtlebot

| Index | Description of Requirement | Result | Comment |
|---|---|---|---|
| FR-01 | The Flying Robot shall move towards a target under control | Not Met | |
| FR-02 | The Flying Robot shall recognize optically an OpenCV's ArUco shape and be able to give information on the location of the target relative to the robot | Met | |
| FR-03 | The Flying Robot shall recognize obstacles in its environment and identify where they are relative to itself | Not Met | |
| FR-04 | The Flying Robot shall be able to make a deviation from its current movement pattern to avoid an obstacle in its path and then return to its pattern | Not Met | |
| FR-05 | The Flying Robot shall be able to identify multiple obstacles and avoid them accordingly | Not Met | |
| PR-01 | The Flying Robot shall be able to identify and locate OpenCV's ArUco shapes within a 15m radius | Not Met | |
| PR-02 | The Flying Robot should be able to process an image and plot obstacles at a rate of 10 Hz | Not Met | |
| PR-03 | The Flying Robot shall be able to navigate an environment and detect up to two obstacles simultaneously | Not Met | |

Table 3: Summary of Results for the Flying Robot

| Index | Description of Requirement | Result | Comment |
|---|---|---|---|
| IT-01 | The Gazebo simulator shall be run on a laptop and robots within the simulation will be interfaced through ROS | Met | |
| IT-02 | Communication to the Turtlebot will be done through ROS over USB | Met | |
| IT-03 | The Flying Robot will be communicating through ROS over WiFi wireless network | Not Met | |
| IT-04 | A simple interface, perhaps command line, will allow a user to select a marker as a target to start the robot's movement towards the target and this signal should be received over wireless transmission | Not Met | |

Table 4: Summary of Results for the Interface Requirements

| Index | Description of Requirement | Result | Comment |
|---|---|---|---|
| SimR-01 | A Gazebo simulation environment which roughly approximates a lab environment with marker placed around will be created | Met- we kinda did this last semester no? | |
| SimR-02 | Stationary obstacles will be added to the lab simulation environment and a Turtlebot shall navigate toward markers in the lab while avoiding obstacles | Not Met | |
| SimR-03 | An environment with obstacles will be created a Flying Robot shall navigate toward markers in the simulation while avoiding obstacles | Not Met | |

Table 5: Summary of Results for the Simulation Requirements

### 6.2 Interface Requirements

### 6.3 Simulation Requirements

### 6.4 Implementation Requirements

### 6.5 Schedule Requirements

## 7 Conclusion

The detailed design has been shown for Computer Optics for Analyzing Trajectories in Mostly Unknown, Navigation Denied, Environments (COATIMUNDE). The background outlined how the project has progressed over the year and the motivation behind it. It gave a summary of the previous Data Item Deliverables, such as the Statement of Requirements and the Preliminary Design Document. The design section laid out the final design of the project and how it was implemented. The verification and validation section explained how different aspects of the project were tested throughout the year. The results section showed the final results and how they proved we met the requirements previously outlined. A discussion section gave a detailed analysis of the design in comparison with the requirements and gave recommendations for future expansions of the project. The final design laid out in this document will

| Index | Description of Requirement | Result | Comment |
|---|---|---|---|
| ImpR-01 | OpenCV programs shall be created that can use a single video stream and identify both markers and obstacles | Met | |
| ImpR-02 | Prior to testing on the Turtlebot, the program shall be implemented on the gazebo simulation | Met??? | |
| ImpR-03 | Usin the robot in the simulation environment the appropriate components, tools, and libraries to interpret and OpenCV stream, make decisions based on the environment, and execute instructions will be developed | Partially Met | Kinda used Gazebo |
| ImpR-04 | The simplest obstacle avoidance algorithm must be implemented on a Turtlebot using ROS | Met | |
| ImpR-05 | The obstacle avoidance algorithm used for a Flying Robot will be implemented in a simulation | Not Met | |

Table 6: Summary of Results for the Implementation Requirements

| Index | Description of Requirement | Result | Comment |
|---|---|---|---|
| SchR-01 | The first Turtlebot simulation shall be able to operate in a Gazebo environment no later than November $5^{th}$ | Not Met | |
| SchR-02 | The first functional prototype shall be a Turtlebot robot capable of positively identifying a marker, moving towards the marker, and avoiding an obstacle placed in its environment no later than December $18^{th}$ | Not Met | |
| SchR-03 | The first functional prototype shall be capable of identifying a marker, moving towards the marker, and avoiding an obstacle placed in its environment no later than February $18^{th}$ | Not Met | |
| SchR-04 | Data Item Descriptions (DID) to be presented are DID-04: Preliminary Design Specification due November $22^{nd}$, DID-05: Preliminary Design Review Presentations due November $29^{th}$, and the DID-06: Schedule Update is due January $17^{th}$. The DID-07: Final Detailed Design Document is due March $21^{st}$, the Final Project Presentation, DID-08, is March $28^{th}$ and the Final Project Demonstration, DID-09, is on April $9^{th}$ | Met | |

Table 7: Summary of Results for the Schedule Requirements

also be used in the the Final Project Presentation and the Final Project Demonstration.

# References


[1] K. Stephan and A. N. Hebb, "Data item deliverable 03- statement of requirements", 2018.

[2] A. J. Barry and R. Tedrake, "Pushbroom stereo for high-speed navigation in cluttered environments", in *Robotics and automation (icra), 2015 ieee international conference on*, IEEE, 2015, pp. 3046–3052.

[3] K. Stephan and A. N. Hebb, "Data item deliverable 04- preliminary design specification", 2018.

[4] ——, "Data item deliverable 06- schedule update", 2019.

[5] *About - opencv library*. [Online]. Available: `https://opencv.org/about.html`.

[6] M. Wise and T. Foote, *Rep: 119 - specification for turtlebot compatible platforms*, Dec. 2011. [Online]. Available: `http://www.ros.org/reps/rep-0119.html`.

[7] *Asctec pelican - uas for computer vision and slam*. [Online]. Available: `http://www.asctec.de/en/uav-uas-drones-rpas-roav/asctec-pelican/`.

[8] Y. K. Hwang and N. Ahuja, "A potential field approach to path planning", *IEEE Transactions on Robotics and Automation*, vol. 8, no. 1, pp. 23–32, 1992.

[9] S. A. Bortoff, "Path planning for uavs", in *American Control Conference, 2000. Proceedings of the 2000*, IEEE, vol. 1, 2000, pp. 364–368.

[10] B. Williams and I. Reid, "On combining visual slam and visual odometry", in *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, IEEE, 2010, pp. 3494–3500.