Tailsitter Localization Using AprilTag and the Kalman Filter for Aerial Docking

by

Abdurrahman Ayyaz Qureshi

Supervisor: Jonathan Kelly
April 2019

This page was intentionally left blank.

University of Toronto

ESC499Y1 Engineering Science Thesis

Tailsitter Localization Using AprilTag and the Kalman Filter for Aerial Docking

Author:

Abdurrahman Qureshi

Supervisor:

Professor Jonathan Kelly

April 2019

Abstract

A tailsitter is a type of air vehicle that is characterized as a VTOL (Vertical Takeoff Landing) and has two flight modes: hover and fixed wing. It is susceptible to disturbances in hover mode, so one conceived solution was to aerially dock two of them to form a quadcopter-like configuration that is inherently more stable and where the motors can be actuated to better reject disturbances. One major requirement for aerial docking is to design and implement a localization method so that two tailsitters can identify each other and maneuver into docking position. This project uses the Hummingbird tailsitter platform developed at the University of Toronto, as well as the visual fiduciary system called AprilTag. AprilTag targets and a camera are added to the Hummingbird gazebo model. Then, a Kalman Filter is implemented to filter the pose measurements. The ability to accurately localize in simulation is shown, and particularly the Kalman filter improves the orientation estimate by cutting down noise by a factor of 2. Next, the mechanical design of the Hummingbird is modified using Solidworks to support an onboard computer, the Odroid XU4, and a camera, the Intel Realsense R200. New parts are 3-D printed/laser-cut and integrated onto the Hummingbird, and the controllers are subsequently re-tuned to achieve a stable hover. A mock tailsitter is then built with Apriltag targets on the side, and localization is attempted. It is shown that the Kalman filter actually degrades the localization estimate. The primary achievements of this thesis are a) adding vision capabilities to the Hummingbird and b) integrated the AprilTag software. By testing localization and filtering on hardware, important insights are gained and next-steps are suggested.

Acknowledgements

First and foremost, I would like to thank Professor Jonathan Kelly for taking me as his thesis student and for his guidance throughout the year. Apart from the technical guidance, I really appreciated his understanding when I failed in assembling the Hummingbird PCB, and also his great sense of humour during our meetings.

Next, I give my thanks to Mike Zhang, Yilun Wu and Jason Wang. Mike Zhang gave up his precious time to help me run multiple experiments and provided thoughtful discussion. Yilun Wu made this thesis project possible and is responsible for creating the Hummingbird platform on which this project builds. Jason Wang helped modify the hummingbird gazebo model and integrate AprilTag.

Lastly, I want to thank the entire Faculty of Applied Science and Engineering, and specifically the Division of Engineering Science, for their tireless efforts in the background to create a fantastic experience and to ensure the future success of all their students.

Contents

List of Figures

1 Introduction

A tailsitter is an air vehicle that is characterized as VTOL (Verticle Take Off Landing). A labelled diagram of the mechanical structure and important components is shown in figure 1. At the top are two propellers connected to two high RPM brushless motors, following by the main body which also acts as wings, and then elevons which are connected to two servos.

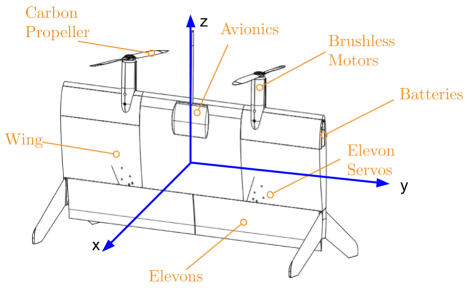
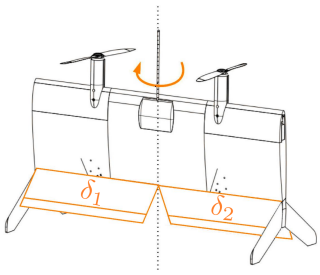
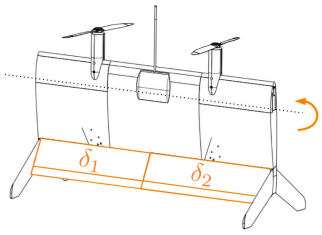


Figure 1: Tailsitter diagram

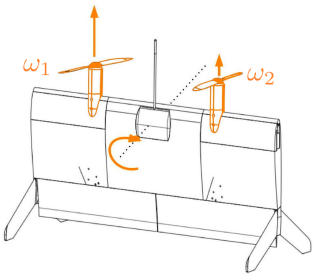
The two motors at the top are used to produce thrust along the body z-axis and to produce a rolling moment by through differential thrust as shown in figure 1(c). The elevons can be used to produce a yaw moment and pitch moment as shown in figure 2(a) and (c) respectively. Once the tailsitter has taken off and is hovering, the elevons can be used to produce a pitch moment and rotate the tailsitter 90 degrees and it flies like a traditional fixed wing aircraft; the two rotors generate forward thrust while airflow around the wings generates upward thrust like that in traditional airplanes.



(a) Body Yaw Actuation



(b) Body Pitch Actuation



(c) Body Roll Actuation

Figure 2: Tailsitter actuation

Tailsitter vehicles combine the advantages of quadcopters and fixed-wing aircrafts. Because tailsitters are a type of VTOL, they do not require long runways which can

be expensive to build and maintain. At the same time, forward flight is much more energy efficient than quadcopters which must run 4 motors at very high speeds all the time. This has lead companies such as Google Wing to experiment with tailsitters for parcel delivery [1]. The inherent tradeoff though is that in hover mode, the tailsitter is susceptible to disturbances about the y-axis. This is easy to see by considering the moment about the y axis, $\tau_y = I_{yy}\alpha_y$. Because the tailsitter is relatively thin, I_{yy} is small, which means any disturbance force will result in a large angular acceleration about the y-axis. Google Wing reportedly cancelled the project due to tailsitter instability [2].

Figure 3: Google Wing reportedly experimented with using tailsitters for parcel delivery. The project was cancelled due to tailsitter instability.

One conceived solution to this problem is aerial-docking. In this context, aerial docking is defined as connecting two tailsitters mid-air to form a rigid-body. Aerial-docking has two benefits. First, it increases the moment of inertia about the y-axis making the system inherently more "stabilizable." Second, the docked formation will have four high rpm motors in a quadcopter-like configuration which it can actuate to better reject disturbances.

The goal of aerial docking can broken down into a loosely ordered series of steps:

1. Design a docking mechanism
2. Design a localization method $\longleftarrow$ (scope of this project)
3. Design distributed control scheme

Figure 4: Concept rendering of final docked configuration.

4. Build two tail sitters

5. Maneuver the tailsitters to docking position and dock

First, some sort of rigid docking/connection mechanism that can be programatically activated needs to be designed. Second, a localization system needs to be designed so that the tailsitters can recognize and maneuver relative to each other to move to docking positions. Third, a distributed control scheme needs to be implemented so that the docked tailsitter configuration can maintain hover flight. Next, two tailsitters need to be built. Finally, all of these systems need to be integrated; the tailsitters need to be launched, localize their positions, maneuver to docking position, dock, and switch to the distributed control scheme.

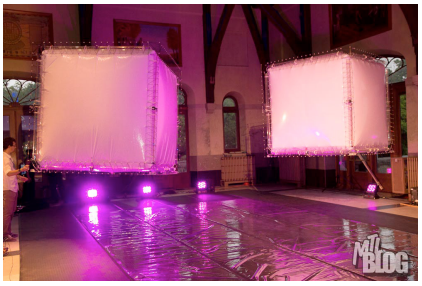
The scope of this project is the design, implementation and simulation of a localization method to achieve goal number 2. The tailsitters should be able to execute relatively complex trajectories from up to 1m away from each other while still maintaining a relatively accurate estimate of each other's position. If this can be accomplished, feasibility to maneuver to docking position is proven.

2 Feasibility of Aerial Docking

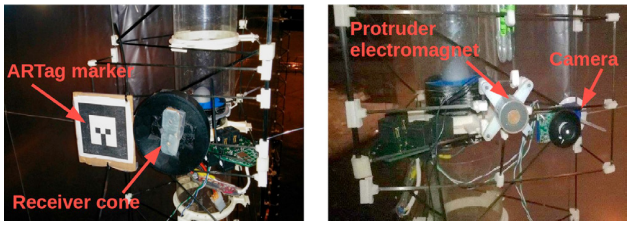
The two big hurdles to aerial docking is the ability to physically maneuver two tailsitters into a docking position, and a control strategy to maintain hover flight in a docked position. To the best of the author's knowledge, two tailsitters have not been

docked in mid-air before, however there have been other docking studies of ground and air vehicles that can be used to as guidelines for this project.

One example of aerial docking done in simulation is of two cubic blimps by Abouzakm et al. [3]. These cubic blimps are referred to as "Tryphons" and their purpose is to produce artistic performances by taking shapes in the air. Abouzakm et al. use a monocular camera and an ARtag marking system for maneuvering and aligning the tryphons, and magnets to physically connect the two blimps in place. Abouzakm et al. breakdown guidance and navigation into two steps: rendezvous and docking. During the rendezvous phase, the two tryphons come into visual proximity via some external global localization scheme so that the attached cameras can detect the ARtags and produce pose estimations. Then, one tryphon, named the "target tryphon," is assumed to remain "in perfect regulation at the desired location throughout the docking process." The other tryphon maneuvers to align the magnets attached to each tryphon and bring them close to each other until the magnetic force becomes great enough to establish a connection.



(a) Tryphon cubic blimps



(b) ARtag marking system targets

Figure 5: Tryphons and ARtag targets used for localization.

A particularly impressive example of distributed control is the distributed flight array developed at ETH Zurich by Oung et al. [4]. A collection of modules having only one rotor and incapable of stable flight themselves, docked together on the ground and achieved a stable hover flight [5]. The pattern of docking was not known beforehand exactly, but the algorithm was able to adapt and achieve a stable hover.

Figure 6: Distributed flight array developed at ETH Zurich by Oung et al.

3 Background

3.1 Hummingbird Platform

This project utilizes the Hummingbird platform which is a tailsitter developed at the University of Toronto by a previous thesis student, Yilun Wu [6]. Hummingbird is built ontop of the PX4 open source flight control software [7].

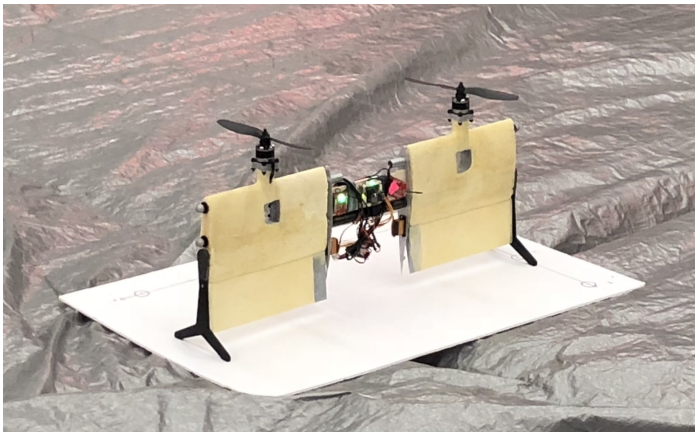


Figure 7: The hummingbird tailsitter

PX4 implements all levels of software from hardware drivers up to the position, attitude, rate cascaded PID control architecture, which has been customized and tailored to the Hummingbird to optimize performance. Details of the hardware and control architecture can be found in the original thesis [6].

For intra-process communication on the embedded computer, PX4 uses the uOrb messaging middleware, which uses a publish-subscribe communication model. To receive external messages from an offboard computer, PX4 uses mavlink, which is a messaging protocol specifically designed for use in drones. In Mavlink, messages are

defined in XML, and C serializers, deserializers and message structures are automatically generated. Once the hummingbird receives a data buffer, it deserializes it into a mavlink message, packs it into a uOrb message, and publishes it to any subscribed processes. The full middleware architecture is shown in figure 8.

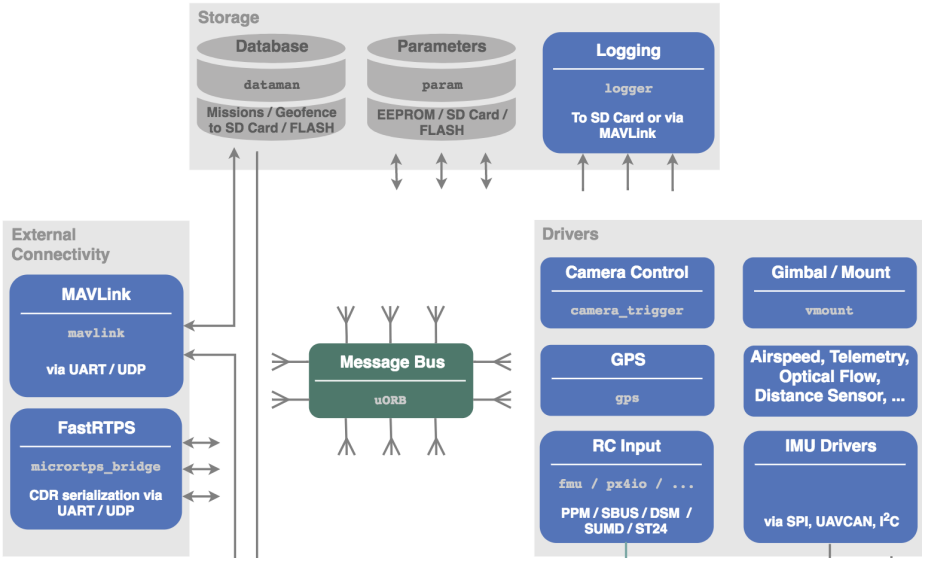


Figure 8: The PX4 embedded middleware architecture.

There are several tools available to send mavlink messages to the hummingbird. In this project, ROS is extensively used, so it is natural to implement a method of converting ROS messages to mavlink messages and sending them automatically. This is implemented in a convenient ROS package called "mavros" [8]. The full communication architecture is shown in figure 9.

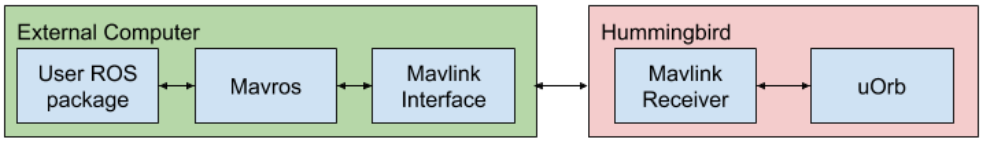


Figure 9: End-to-End Communication Architecture from user level ROS node running on an offboard computer to uOrb.

PX4 also comes built with support for a high fidelity Software-In-The-Loop (SITL) simulation for fast prototyping and regression testing. One of the simulators it supports is Gazebo, which has a robust physics engine and can be used to accurately

model robot dynamics and complicated environments. As part of the original thesis, a gazebo plugin was written that implements a theoretical model of the dynamics with experimentally determined parameters [6]. The PX4 tailsitter model is shown in figure 10.

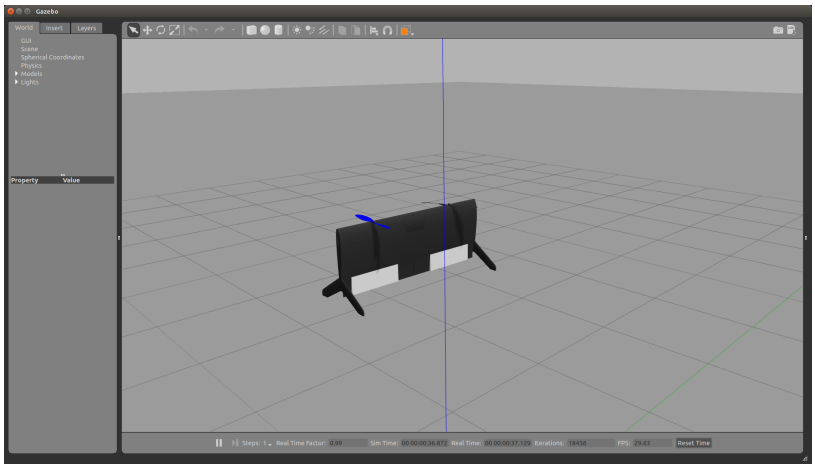
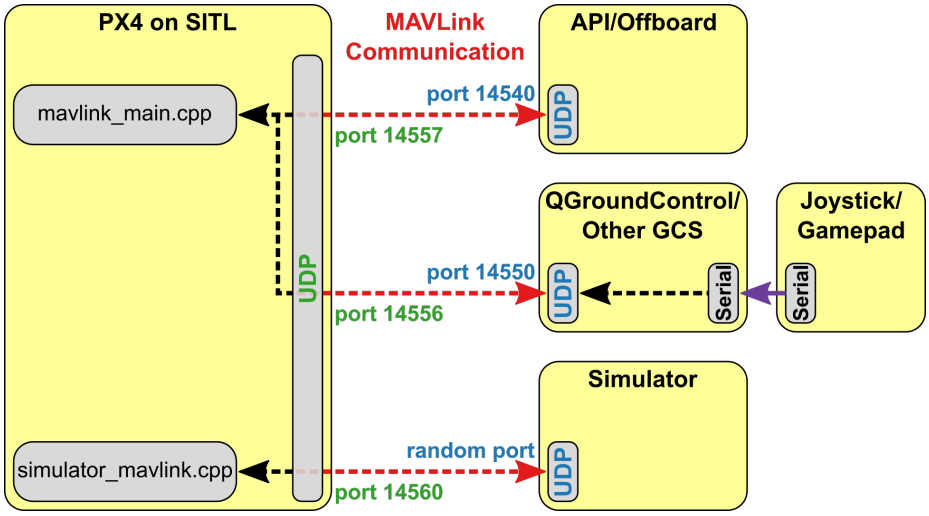


Figure 10: The PX4 tailsitter model for gazebo.

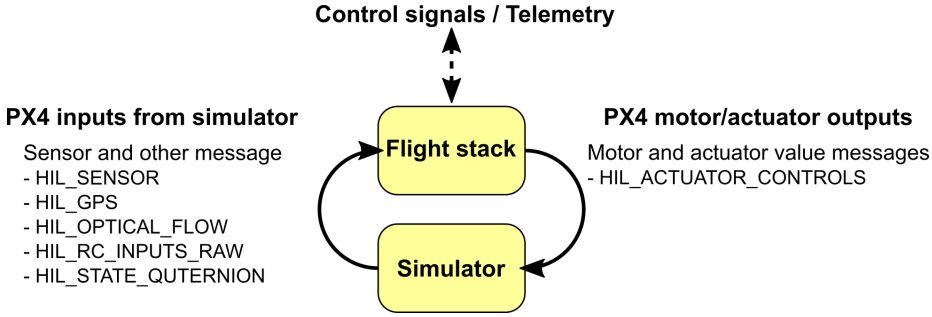
When the simulation is started, PX4 is loaded with "simulator_mavlink.cpp" which sends actuator inputs to the simulation and receives sensor and state data from the simulation over UDP port 14560. This is shown in figure 11.

3.2 Visual Fiduciary Systems

This section summarizes three prevalent visual fiduciary systems, which are systems where an object is placed in a camera image for use as a point of reference of measurement. Originally, visual fiduciary systems were developed for Augmented Reality (AR) applications, but since then they've been applied to many areas including calibration, robotic docking, and more. This section compares AprilTag, ARTag and CALTag. In each of these systems, "targets" are placed on objects that need to be tracked. Given a camera image of the target and the physical dimensions of the target, software can compute the position and orientation of the camera viewing the target, relative to the target. These fiduciary systems are also very low-cost localization solutions because targets can be printed from ordinary printers.



(a) The PX4 SITL architecture overview.



(b) Messages exchanged between Gazebo and PX4.

Figure 11: PX4 Gazebo Intercommunication

3.2.1 ARTag

ARTag was developed by Mark Fiala and it is a successor to the out-dated ARToolKit [9]. To understand ARTag, it is useful to start with ARToolKit. ARToolKit targets had arbitrary images inside black boxes which were matched against a database at runtime. As the database grew, computation costs increased and false matches increased. ARTag switches to 2D binary barcode patterns which encodes a bit code (e.g. 16 bits) so that an image can be decoded and matched must faster to the known deployed bit codes. In practice, while decoding an image to the bit code, a few bits are incorrect so that the code doesn't match exactly, but if the number of incorrect bits is within some tolerance (typically up to 2), the algorithm outputs a positive match. An ARToolKit target and ARTag target is shown in figure 12.

(a) ARToolKit (b) ARTag

Figure 12: ARToolkit target and an ARTag target

3.2.2 AprilTag

AprilTag is a visual fiducial system originally developed at the University of Michigan Robotics Laboratory by John Wang and Professor Edwin Olson [10] [11]. AprilTag builds on ARTag and it has gone through two iterations: AprilTag1 and AprilTag2. AprilTag1 introduced a more robust quad detection algorithm by using low pass filtering, image gradients, and line fitting to find squares in an image. It's other major change improvement over ARTag was the use of a lexicode system to ensure that deployed bit codes are not too similar to each other, and even to itself under 90, 180, and 270 degree rotations. The measure of similarity used is "hamming

distance" which counts the number of positions that are different between two equal length strings. For example, if two codes are only different in one bit, their hamming distance is one. If the number of allowable bit errors is 1 or 2, this could lead to a false detection. AprilTag is highly resilient to these types of false positives. An AprilTag target is shown in figure 13.

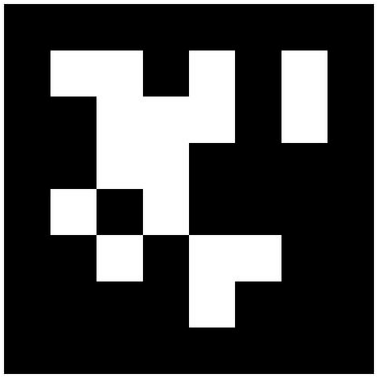


Figure 13: AprilTag target

AprilTag2 improves on AprilTag1 algorithmically. From user feedback, the authors of AprilTag1 understood that tag occlusion was not a significant concern and in practice, most users don't allow any bit errors anyway. At the cost of less robustness to occlusion, AprilTag2 is computationally much faster due to a faster tag boundary segmentation method and much faster decoding and matching.

3.2.3 CALTag

CALTag (CALibration Tag) was developed at the University of British Columbia by Atcheson et al. specifically for camera calibration purposes [12]. CALTag has an added capability of error detection (but not error correction) by including a checksum as part of the bit code that is encoded into the 2D barcode pattern. In practice, for reasonable sized targets, this severely reduces the number of available targets that can be generated. The targets are generated by creating an MxN grid similar to a checkerboard where each cell has a strictly white or black border around it. The

CALTag software decodes it in the four possible orientation and uses the checksum to verify the correct orientation. An example is shown in figure 14.

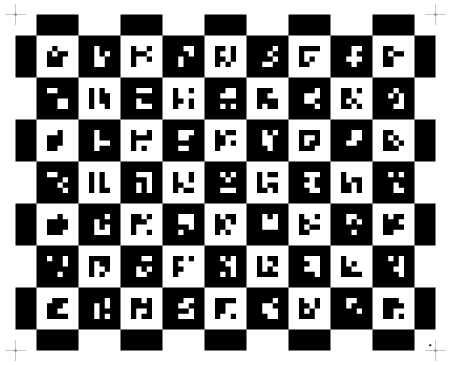


Figure 14: CALTag target

The major advantage of CALTag is that by creating these large grids, there are many more calibration points (the corners of the checkerboard) that can be used as reference points.

3.2.4 ARTag vs AprilTag vs CALTag

AprilTag2 is a newer technology that directly builds on top of and has a lot of advantages over ARTag. Sagitov et al. directly compared the three in a number of configurations including partial occlusion, and rotations about the three axises (see figure 15) [13]. AprilTag2 and CALTag were robust against all normal and lateral rotations up to 65 degrees, while some ARTag targets failed at rotations greater than 10 degrees. AprilTag and ARTag were both sensitive to occlusion while CALTag was robust.

But while CALTag outperforms AprilTags in detection on occluded targets, occlusion is not a big concern in this project. CALTag is also quite a lot slower than AprilTag2. It's benchmark was on a minimum of 2 megapixel images on which it took approximately 2 seconds, which corresponds to a rate of 0.5Hz. On the other hands, without modification, AprilTag2 outputs detections at a rate of 3Hz. Another advantage of AprilTag is that a ROS wrapper around the AprilTag software is readily available as a package online [14] [15]. Given camera information and camera images,

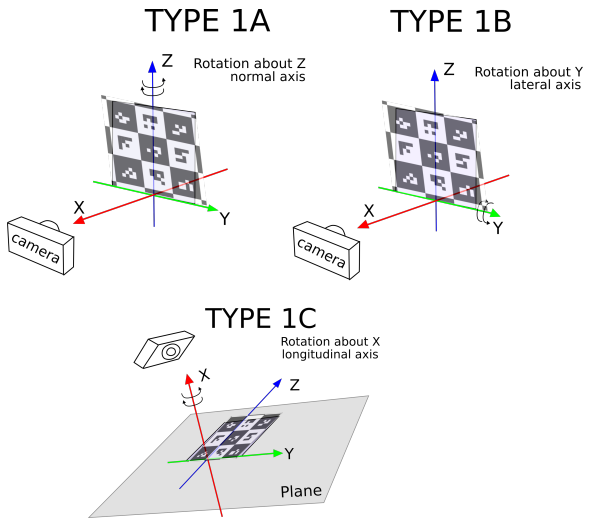


Figure 15: Rotations tested by Sagitov et al.

tag detections are published as transforms on the /tf topic. This allows the ability to exploit the ROS ecosystem.

3.3 Kalman Filter

The kalman filter is a recursive filter used to calculate optimal state estimates of a linear system by combining model estimates with output measurements, both containing stochastic, zero-mean gaussian noise [16]. The extended kalman filter generalizes this for non-linear systems by repeatedly linearizing the system at previous state estimates to estimate model noise propogation. It is not guaranteed to be optimal, but in practice, it is highly effective and a de-facto industry standard filter.

3.3.1 Extended Kalman Filter

Assume a non-linear system is given by the following set of equations:

$$x_k = f(x_{k-1}, u_{k-1}) + w_{k-1}$$
$$z_k = h(x_k) + v_k$$

where x_k is the system state, u_k is the system input, w_k and v_k are zero-mean gaussian process noise and measurement noise vectors respectively, and z_k is the measurement.

It is assumed the process and measurement noise are uncorrelated and their covariance matrices are given by the following:

$$Q = E[w_k w_k^T]$$
$$R = E[v_k v_k^T]$$

Let the state estimates be given by $\hat{x}_k$ and define a state error covariance matrix:

$$P_k = E[(x_k - \hat{x}_k)(x_k - \hat{x}_k)^T]$$

The filter is initialized with some initial state estimate $\hat{x}_0$ and initial state error covariance matrix P_0. It can be shown that the optimal state estimate at step k is given by the following equation:

$$\hat{x}_k = \hat{x}_k^f + K(z_k - h(\hat{x}_k^f))$$

where:

$$\hat{x}_k^f = model\ prediction$$
$$K = kalman\ gain$$

It is interesting to notice that this is a balance between the model prediction and the error between the model prediction and the measurement. The second term $K(z_k - h(\hat{x}_k^f))$ is called the *innovation* term. The optimal model prediction is given by the following equation:

$$\hat{x}_k^f = f(x_{k-1}, u_{k-1})$$

And the kalman gain is calculated using the following equation:

$$K = P_k^f H^T (H P_k^f H^T + R)^{-1}$$

The P_k^f matrix is defined as the prediction error covariance matrix and H is the linearization of the measurement model about the state predicted by the model:

$$P_k^f = E[(x_k - \hat{x}_k^f)(x_k - \hat{x}_k^f)^T] = \Phi P_{k-1} \Phi^T + Q$$
$$H = \frac{dh}{dx}|_{\hat{x}_k^f}$$

where Φ is the error propogation matrix:

$$\Phi = \frac{df}{dx}|_{\hat{x}_{k-1}}$$

Finally, the current state error covariance matrix is computed to be used during the next estimation:

$$P_k = (I - KH)P_k^f$$

3.3.2 Kalman Filter Non-Linear Process/Measurement Noise

Suppose the system model was given by the following equation:

$$x_k = f(x_{k-1}, u_{k-1}, w_{k-1})$$
$$z_k = h(x_k, v_{k-1})$$

Linearizing both models about zero noise:

$$x_k \approx f(x_{k-1}, \hat{u}_{k-1}, 0) + \frac{df}{dw} w_{k-1}$$
$$z_k \approx h(x_k, 0) + \frac{dh}{dv} v_k$$

This is the same set of equations as described above except the process noise and measurement noise is given by $\frac{df}{dw} w_{k-1}$ and $\frac{dh}{dv} v_k$ respectively with the following new

covariances:

$$Q = (\frac{df}{dw})E[w_k w_k^T](\frac{df}{dw})^T$$
$$R = (\frac{dh}{dv})E[v_k v_k^T](\frac{dh}{dv})^T$$

3.3.3 Kalman Filter Input Noise

Suppose the true input to the system $u = \hat{u} + n$ where $\hat{u}$ is the desired/observed input and n is random gaussian noise $n \sim \mathcal{N}(0, \Sigma_u)$. Then:

$$x_k = f(x_{k-1}, u_{k-1}) = f(x_{k-1}, \hat{u}_{k-1} + n)$$

Linearizing about $\hat{u}_{k-1}$:

$$x_k \approx f(x_{k-1}, \hat{u}_{k-1}) + \frac{df}{du}|_{\hat{u}_{k-1}} n$$

Defining $G_k = \frac{df}{du}|_{\hat{u}_{k-1}}$, the equation above becomes:

$$x_k \approx f(x_{k-1}, \hat{u}_{k-1}) + G_k n$$

$G_k n$ is now zero-mean gaussian process noise. The covariance matrix of $G_k n$ is given by the following:

$$E[(G_k n)(G_k n)^T] = G_k \Sigma_u G_k^T$$

3.3.4 Adaptive Kalman Filter

The EKF assumes constant measurement noise covariance matrix R but in many applications the measurement noise often varies over time. In response, many adaptive kalman filters have been developed which automatically update R based on a window of past model predictions and measurements. Almagbile et al. employ the following "last m-residuals" (i.e. difference between estimates and actual) based update

equations [17]:

$$\hat{R}_k = B_k + CP_kC^T$$
$$\bar{v}_k = z_k - C\hat{x}_k$$
$$B_k = \frac{1}{m}\sum_{i=1}^{m} v_{k-i}v_{k-i}^T$$

This method seems promising for AprilTag related noise measurements where empirically the noise grows as the partner tailsitter drifts further away or is not directly facing the camera.

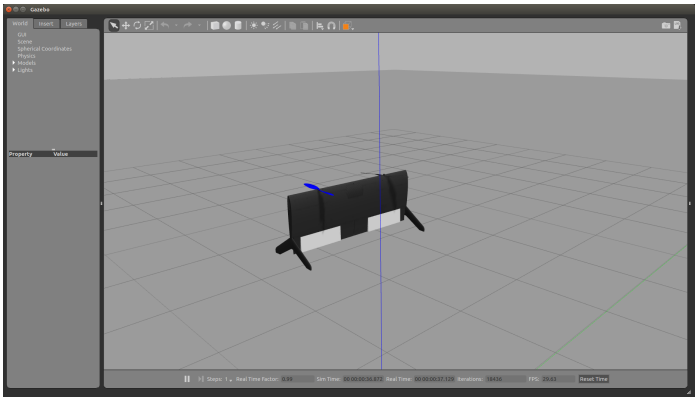
4 Method

This section describes how the simulation capabilities were enhanced to support a high fidelity simulation of localization and maneuvering. Then, it describes how the tailsitter hardware was modified to support an onboard computer to run the localization software.

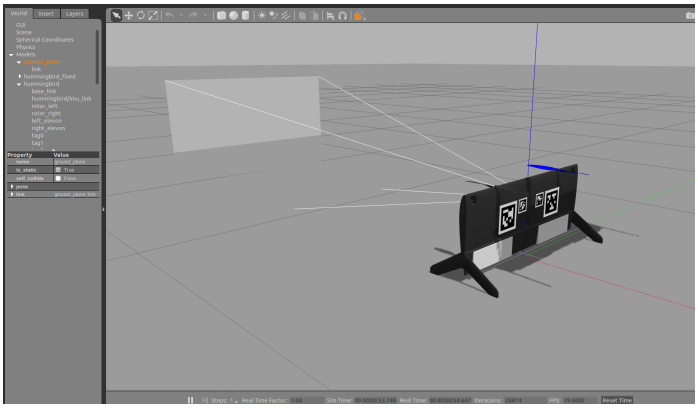
4.1 Integrating a Visual Fiduciary System: AprilTag

AprilTag was the chosen visual fiduciary system due to it's fast speed and convenient ROS wrapper. The first step to enabling the simulation was adding AprilTag targets to the tailsitter gazebo model as well as a camera for taking images and computing pose measurement. This was done with the help of UTIAS summer student Jason Wang. Four tags from the "tag36h11" family were chosen (36 bits, hamming distance 11). Two large targets (10cm X 10cm) and two small targets (5cm x 5cm) were added to the front-facing side of the wings. They were arranged such that as the distances between the viewing camera and the tailsitter decreases (e.g. the two tailsitters get closer together), the large targets will not fit in the image frame and the pose estimation can switch to using the small targets. A picture of the old model and new model is shown in figure 16. Finally, a ROS node was written that publishes

the tailsitter to camera transform to the /tf topic, which means measurements can be directly converted to the tailsitter frame with the tf2 ROS package.



(a) Original tailsitter model.



(b) New tailsitter model with AprilTag targets and a camera.

Figure 16: Tailsitter gazebo model before and after modification

Next, the "apriltags2_ros" ROS package was downloaded and the launch file was modified to launch the apriltags detection nodes, while feeding in the correct topics containing camera information and camera images. The end result if this wiring is shown in figure 17. On the left, one can see the tag detections, including their position and orientation relative to the base_link tailsitter, in RVIZ.

4.2 Implementing Kalman Filtering

In general, the measurements were very highly accurate when the tailsitters were $< 30cm$ apart and the tailsitters were directly facing each other, but any further, and although the targets were always detected, the measurements became noisy,

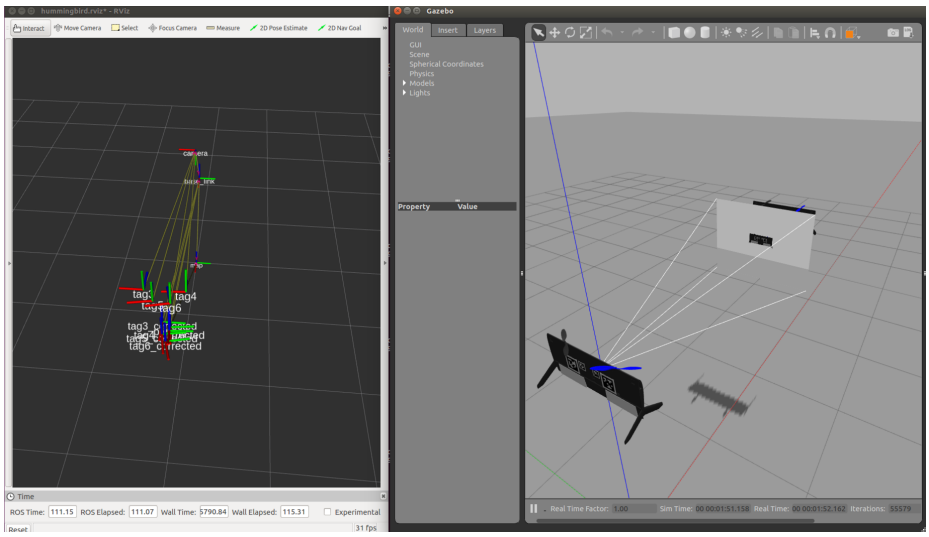


Figure 17: (Left) RVIZ showing tag pose detections. (Right) Gazebo simulation.

particularly when the tag was rotated about its normal and lateral axis. Ideally any global rendezvous system does not have to precisely position the tailsitters to such a close proximity. Additionally, when the tailsitter was maneuvering, the measurements became noisy. A third party Kalman filter implementation on Github written by Markus Herb was used to filter AprilTag measurements [18]. The original kalman filter implementation was modified to propagate noise in the system input, and to take a *dt* term when computing the model prediction. The process and measurement models are desribed in section 5. The noise covariance matrices are enumerated in appendix A.

4.3 Improving the Hummingbird Platform Software Suite

Although unrelated to the primary goals of this thesis, the Hummingbird platform was improved to add functionality, remove bugs, and reduce complexity, some of which is talked about here.

To enable more complicated trajectories to create more interesting and difficult test cases to test the robustness of the estimation, the PX4 "ts_path_planner" module was modified to include a "raw" mode. Previously, "ts_path_planner" generated a linear path from the start position to the end position, creating it's own velocity pro-

file, and publishing intermediate setpoints. In raw mode, setpoints published through ROS on the "/mavros/setpoint_raw/local" topic are sent and executed directly and without modification. MATLAB and the ROS interface package was used to generate and publish setpoints to track a circle [19].

Previously, a software application called QGroundControl (QGC) was used to connect a game controller to the Hummingbird vehicle to issue commands [20]. QGC also provided visualizations of live-streamed data, but in practice that feature was not used. It was a buggy software that had to be independently compiled and run. QGC was removed and a custom ROS node was built that takes joystick input and sends commands such as arm, takeoff, land and disarm to the Hummingbird vehicle.

4.4 Mechanical Modification of the Hummingbird

To enable vision, an onboard computer and camera needed to be added to the hummingbird tailsitter. The mechanical design of the Hummingbird was therefore modified in Solidworks to add mounting points for the chosen onboard computer and camera. The Odroid XU4 was chosen as the onboard computer because of its lightweight and relatively strong processor. The Intel Realsense R200 camera was chosen because of its light weight and large developer community. The inner ribs of the tailsitter were modified to add a bottom support beam, and the original mounting plate was lengthened to make room for a camera. The Solidworks assembly of the modified Hummingbird center is shown in figure 18. The Pololu 5V 4A D24V50F5 regulator was used to power the Odroid from the battery. The new parts were 3D printed/laser-cut/ordered and integrated and the result is shown in figure 19. Finally, the controllers were retuned to achieve achieve stable hover; the gains are listed in table 1.

A "mock tailsitter" was also built with AprilTags in the same configuration as the simulated partner tailsitter as well as vicon markers for benchmarking. This mock tailsitter is shown in figure 20.

Gain	$\zeta_{p,xyz}$	$\tau_{p,xy}$	$\tau_{p,z}$	$k_{\omega,pitch}$	$k_{\omega,roll}$	$k_{\omega,yaw}$	$\tau_{\omega,pitch}$	$\tau_{\omega,roll}$	$\tau_{\omega,yaw}$
Value	1.4	0.6	0.4	3	5	12.5	0.06	0.15	0.044

Table 1: New controller gains.

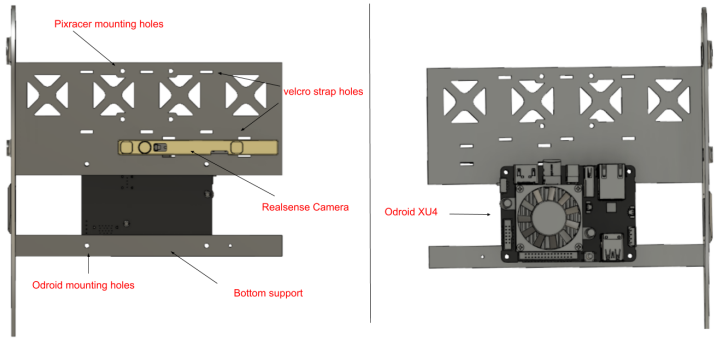


Figure 18: Front and back views of the Solidworks assembly of modified Hummingbird center.

(a) Front. (b) Back.

Figure 19: Modified mechanical design of tailsitter with mounted Odroid XU4 and realsense camera.

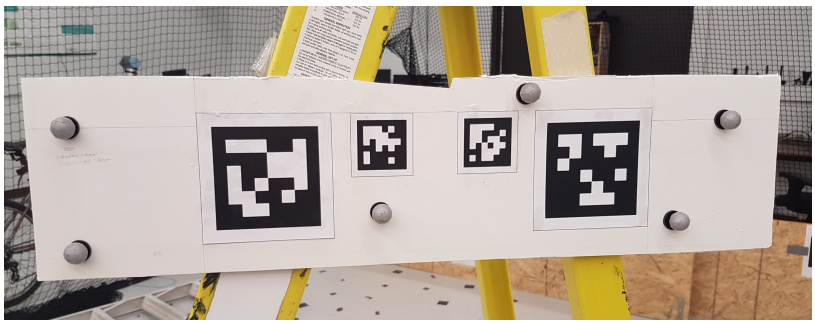


Figure 20: Mock tailsitter to test localization estimation.

5 System Modelling

It is assumed that one tailsitter will maintain a hover flight while the other maneuvers to perform docking. The tailsitter that is viewing and estimating the position of the other tailsitter is named "base_link" and the other tailsitter "partner." The base_link and partner are modelled as point masses and assigned frames 1 and 0 respectively. The setup is shown in figure 21. The partner frame is modelled as an inertial frame, e.g. as having 0 linear and angular velocity and the base_link frame with some *relative* velocity v, acceleration a and angular velocity ω. The goal is to track frame 0 with respect to frame 1.

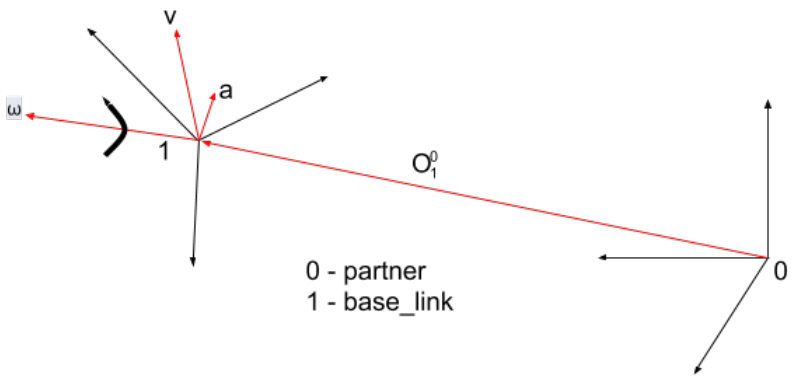


Figure 21: Point mass model of the partner and base_link tailsitters. The partner tailsitter is modelled as an inertial frame. The gravity vector is assumed to point in the direction of $-z_0$.

5.1 System Parameterization

Define a state variable x:

$$x = \begin{bmatrix} o_0^1 \\ v \\ \eta \\ b_a \end{bmatrix} \tag{1}$$

where o_0^1 is the position of frame 0 with respect to frame 1, v is the velocity of frame 1 with respect to frame 1, $\eta \doteq [\phi, \theta, \psi]$ is the roll, pitch, yaw of frame 1 with respect to frame 0, and b_a is the IMU accelerometer bias estimate.

Next, define a system input u:

$$u = \begin{bmatrix} a \\ \omega \end{bmatrix} \tag{2}$$

where $a = a_1^1$ is the acceleration of frame 1 with respect to frame 1 and $\omega = \omega_1^1$ is the angular velocity of frame 1 with respect to frame 1.

5.2 Process Model

Begin by summing the vectors from frame 0 to frame 1 and frame 1 to frame 0 which must add up to 0:

$$o_1^0 + R_1^0 o_0^1 = 0$$
$$\dot{o_1^0} + \dot{R_1^0} o_0^1 + R_1^0 \dot{o_0^1} = 0$$
$$\dot{o_0^1} = R_0^1 S(\omega_1^0) o_1^0 - R_0^1 \dot{o_1^0}$$

Identifying $v = v_1^1 = R_0^1 \dot{o_1^0}$ as the velocity of the base_link in its own frame and $w_1^0 = R_1^0 w_1^1$ the above equation becomes:

$$\dot{o_0^1} = -S(\omega_1^1) o_0^1 - v + n_o \tag{3}$$

where an empirically set gaussian noise vector, n_o, with covariance Σ_o is added at the end. This equation governs the dynamics of linear motion of frame 0 with respect to frame 1. The dynamics of η is derived next. The rate of change of the roll, pitch, and yaw as a function of frame 1's angular rate is described by the following equation:

$$\omega = \begin{bmatrix} u \\ v \\ w \end{bmatrix} = \begin{bmatrix} \dot{\phi} \\ 0 \\ 0 \end{bmatrix} + (R_{x,\phi})^T \begin{bmatrix} 0 \\ \dot{\theta} \\ 0 \end{bmatrix} + (R_{y,\theta} R_{x,\phi})^T \begin{bmatrix} 0 \\ 0 \\ \dot{\psi} \end{bmatrix}$$

Rearranging for $\dot{\eta}$:

$$\dot{\eta} = \begin{bmatrix} \frac{u\cos\theta + w\cos\phi\sin\theta + v\sin\theta\sin\phi}{\cos\theta} \\ v\cos\phi - w\sin\phi \\ \frac{w\cos\phi + v\sin\phi}{\cos\theta} \end{bmatrix} + n_\eta \tag{4}$$

where an emperically set gaussian noise vector n_η with covariance Σ_η is added at the end. The derivative of velocity is simply the acceleration:

$$\dot{v} = a_1^1 + n_v \tag{5}$$

where n_v is an empirically set gaussian noise vector with covariance Σ_v. Finally, the IMU accelerometer bias is modelled as a random walk following Kelly et al. [21]:

$$\dot{b_a} = n_{aw} \tag{6}$$

where n_{aw} is a gaussian random variable with mean 0 and some covariance Σ_{aw}.

The measured IMU angular velocity and angular acceleration are:

$$\omega_m = \omega_1^1 + n_\omega \tag{7}$$

$$a_m = a_1^1 - R_0^1\vec{g} + b_a + n_a \tag{8}$$

where n_ω and n_a is zero-mean gaussian noise with covariances Σ_ω and Σ_a respectively, and $\vec{g} = \begin{bmatrix} 0 & 0 & g \end{bmatrix}$ is the gravity vector. Note that any gyroscope bias is ignored.

This fully describes the continuous time process model. Below, the relevant jacobians are provided for completeness.

The derivatives with respect to the state are as follows:

$$\frac{df}{dx} = \begin{bmatrix} -S(\omega_1^1) & -I_{3x3} & 0_{3x3} & 0_{3x3} \\ 0_{3x3} & 0_{3x3} & \frac{d\dot{v}}{d\eta} & 0_{3x3} \\ 0_{3x3} & 0_{3x3} & \frac{d\dot{\eta}}{d\eta} & 0_{3x3} \\ 0_{3x3} & 0_{3x3} & 0_{3x3} & 0_{3x3} \end{bmatrix}$$

where:

$$\frac{d\dot{v}}{d\eta} = \begin{bmatrix} 0 & -gc(\theta) & 0 \\ gc(\theta)c(\phi) & -gs(\theta)s(\phi) & 0 \\ -gc(\theta)s(\phi) & -gs(\theta)c(\phi) & 0 \end{bmatrix}, \frac{d\dot{\eta}}{d\eta} = \begin{bmatrix} \frac{s(\theta)(vc(\phi)-ws(\phi))}{c(\theta)} & \frac{wc(\phi)+vs(\phi)}{c(\theta)^2} & 0 \\ -wc(\phi)-vs(\phi) & 0 & 0 \\ \frac{vc(\phi)-ws(\phi)}{c(\theta)} & \frac{s(\theta)(wc(\phi)+vs(\phi))}{c(\theta)^2} & 0 \end{bmatrix}$$

The derivatives with respect to the inputs are as follows:

$$\frac{df}{du} = \begin{bmatrix} 0_{3x3} & \frac{d\dot{o_0^1}}{d\omega} \\ I_{3x3} & 0_{3x3} \\ 0_{3x3} & \frac{d\dot{\eta}}{d\omega} \\ 0_{3x3} & 0_{3x3} \end{bmatrix}$$

where:

$$\frac{d\dot{o_0^1}}{d\omega} = \begin{bmatrix} 0 & -(o_0^1)_z & (o_0^1)_y \\ (o_0^1)_z & 0 & -(o_0^1)_x \\ -(o_0^1)_y & -(o_0^1)_x & 0 \end{bmatrix}, \frac{d\dot{\eta}}{d\omega} = \begin{bmatrix} 1 & \frac{s(\theta)s(\phi)}{c(\theta)} & \frac{c(\phi)s(\theta)}{c(\theta)} \\ 0 & c(\phi) & -s(\phi) \\ 0 & \frac{s(\phi)}{c(\theta)} & \frac{c(\phi)}{c(\theta)} \end{bmatrix}$$

5.3 Measurement Model

A measurement starts with a camera image of a tag. The camera is mounted on the base_link tailsitter and the tag is mounted on the partner tailsitter. Assign frame 2 to the tag and frame 3 to the camera. This is shown in figure 22. It is assumed that the camera to base_link transformation is known (o_3^1 and R_3^1) and the tag to partner

transformation is known (o_2^0 and R_2^0). The output of the AprilTags software is the camera to tag transformation, or o_2^3 and R_2^3. Then the following equations define the relationship between the measurement and state:

$$R_0^1 = R_3^1 R_2^3 R_0^2 + n_{\eta m}$$
$$o_0^1 = o_3^1 + R_3^1 o_2^3 + (-R_0^1 o_2^0) + n_{om}$$

where $n_{\eta m}$ and n_{om} are gaussian noise vectors added to the measured values of the orientation and position respectively with covariance matrices $\Sigma_{\eta m}$ and Σ_{om}.

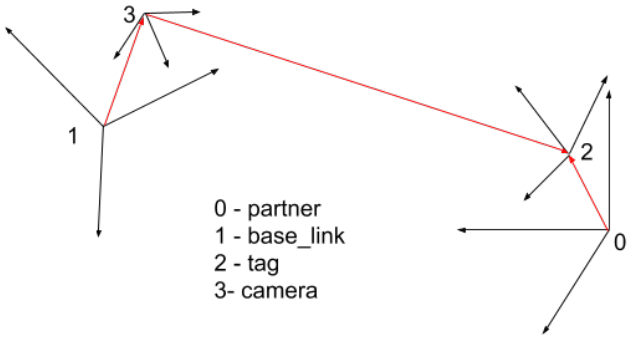


Figure 22: Tag Measurement

6 Results

Localization was tested in simulation by hovering the tailsitter approximately 1.5m away, moving forward and stabilizing at a hover position approximately 0.5m distance, and then moving back. The pose estimate graph results can be found in appendix B.

Localization was also tested on hardware by hovering the tailsitter 0.5m away and 1m away. The pose estimate graph results can be found in appendix C and D respectively. One important observation of the differences between hardware and simulation is that on hardware there is substantial time delay of approximately 350ms between ground truth pose of the partner and the measurement by Apriltag. Nevertheless, ignoring the time delay and using vicon as ground truth to quantify accuracy, tables 2 and 3 enumerate the largest errors of the raw tag measurements and the filter

	Apriltag Measurement Error	Filter Estimate Error
x	3cm	26cm
y	1cm	9cm
z	2cm	2cm
roll	0.03rad	0.10rad
pitch	0.05rad	0.10rad
yaw	0.06rad	0.08rad

Table 2: Largest errors during 0.5m hover.

	Apriltag Measurement Error	Filter Estimate Error
x	5cm	16cm
y	4cm	8cm
z	4cm	7cm
roll	0.03rad	0.09rad
pitch	0.10rad	0.20rad
yaw	0.10rad	0.15rad

Table 3: Largest errors during 1m hover.

output (outlier measurements are qualitatively identified and ignored though). It is important to note that the biases between the stationary Apriltag measurement and vicon measurement was not taken into account. In this respect, the measurements are actually better than what is reported in the table, and qualitatively one can observe the profile over time of the Apriltag measurements mirrors the Vicon measurements well. Instead, the important thing to note is that a) as expected, the errors grow as distance increases and b) the filter makes the estimate worse. Finally, the tables don't portray the existence of outliers which are particularly prevalent in the pitch measurement which we qualitatively explain here. At 0.5m, there were noise spikes of about 0.15rad magnitude in pitch, and at 1m, the magnitude of noise spikes was about 0.3rad.

Ignoring time delays, on hardware, the position of the partner was accurately estimated to within $\sim 3cm^3$ radius sphere at 0.5m and $\sim 7cm^3$ radius at 1m. It would likely be even more accurate if a calibration step to remove the bias between the measurement and ground truth is implemented first.

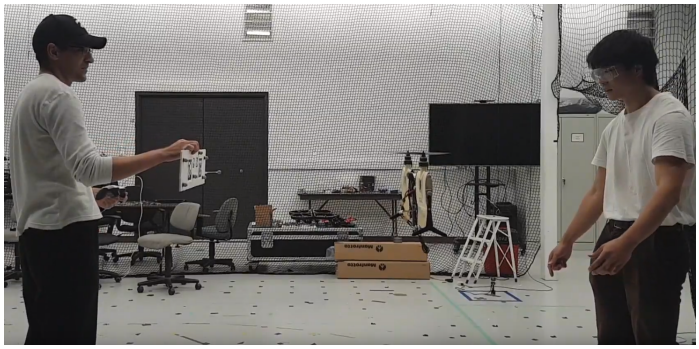


Figure 23: Taking pose measurements while the tailsitter hovers infront.

7 Discussion

7.1 Performance in Simulation

The filter position estimate was largely the same as the measurement due to a high accelerometer noise covariance matrix, but the filter did add high frequency noise. The filter's largest benefit was improving the orientation estimate by cutting down the process noise by a factor of about two. For example, during hover at 1m, the pitch estimate went from +/- 0.04 radians to +/- 0.02 radians. At the same time, it was able to follow the pitch profile as the tailsitter maneuvered toward and away from the partner tailsitter.

7.2 Performance on Hardware

On hardware, in all experiments, the quality of the measurements degraded substantially. Firstly, the AprilTags software output at 24Hz instead of the camera image rate of 30hz. The Kalman filter not only did not improve the pose estimate, but degraded it further. It worsened the time delay, it added extremely bad transients during estimate convergence, it added high frequency noise in the position estimate, and it worsened the orientation estimate. It's only positive effect was to smooth the orientation estimates and mitigate the effects of noise spikes. For example, at 1m, a large noise spike at about 34s can be seen, suggesting the relative pitch at about 0.35 radians or 20 degrees, but the filter rejected that.

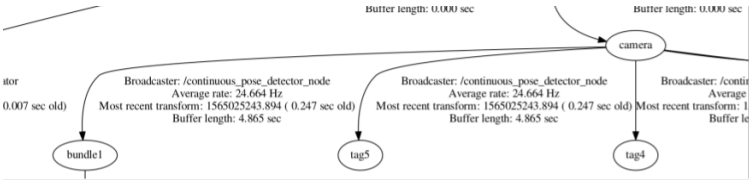


Figure 24: TF2 frames tree visualization. camera to bundle1 is output at 24Hz.

7.3 Model Deficiencies

This section attempts to explain some reasons for the failure of the filter to improve the estimate by analyzing where the model was deficient.

One possible explanation for the high frequency position noise can be intuitively understood by understanding how the model would propagate state based on consecutive measurements of a tailsitter when it is moving in some direction while accelerating in the opposite direction. A measurement will lead the filter to infer a certain velocity, which will cause the model to predict it is continuing along the same path. The accelerometer noise is so high that it might as well be ignored. Subsequent measurements will then conflict with model predictions.

Another possible reason stems from the empirically observed fast divergence of the estimate in the absence of measurements (within 500ms). There could be a few reasons for this. The high accelerometer noise leads to random velocity estimates. A more likely explanation is the slight accelerometer bias (even though it is being estimated) builds up in the velocity state.

Although it was not explored in this thesis, another big source of model error will be when the partner tailsitter is not truly static and upright, but another tailsitter with non-zero acceleration/velocity and a non-upright orientation.

Finally, another weakness of the model is constraints imposed by the use of euler angles. The partner needs to be in an orientation that does not have discontinuities nearby, because the EKF assumes continuous random variables; otherwise it will give wildly incorrect estimates. For example, if the partner is yawed approximately 180 degrees relative to the base_link tailsitter, measurements of yaw will flip between pi and -pi.

7.4 Next Steps

This section lists a set of recommendations for next steps to achieve docking.

- **Replace EKF with a LPF**.

  One could attempt to re-tune the EKF's noise covariance matrices (e.g. increasing the covariance of the velocity estimate) but as described above, the EKF model has fundamental flaws. Additionally, in practice, even on hardware, the raw apriltag measurements are actually quite good. One should remove the EKF and instead add a low-pass filter.

- **Make special considerations for the existence of a time delay in measurements**.

- **Add C.O.M. offset to model**

  In practice, non-negligible x and y position biases exist because the C.O.M. is not perfectly symmetrical. A significant I term in the controllers adds transients. Ensuring symmetry during mechanical design adds significant constraints to the design and cannot be perfectly achieved anyway.

- **Remove vicon input to the tailsitter and attempt to hover relative to tag position**.

  The PID control relies heavily on the accuracy of the velocity measurement to dampen oscillations. Either a new control system must be implemented, or velocity needs to be accurately estimated during hover. An intermediate step might be to reduce the vicon measurement rate and add a delay to test control. It will also involve ensuring accuracy of model parameters so a significant x, y, and z bias don't exist so that the tags stay in view of the camera.

- **Change the Intel Realsense R200 Camera**

  The Intel Realsense R200 camera often crashed due to USB powering issues from the Odroid. Other lightweight cameras should be tested for reliability and used instead.

8 Conclusion

In this thesis, localization capabilities were added to the Hummingbird tailsitter. First, a high fidelity simulation using gazebo was developed by updating the PX4 tailsitter model with cameras and AprilTag targets. Next, the apriltags2_ros ROS package was integrated to capture pose measurements of tags using the camera and measurements of a partner tailsitter were calculated. Then, the pose measurements were filtered using an EKF. It was shown that the EKF improves the pose measurements in simulations. Then, the Hummingbird was mechanically redesigned to add an onboard computer and camera. The controllers were re-tuned and localization was tested again. It was shown that the measurements become noisier and the EKF actually further degraded the estimate, but that by-and-large, the raw measurements were still fairly good. The partner tailsitter's position was localized to about $3cm^3$ radius sphere at about 0.5m and $7cm^3$ radius at about 1m. It is likely that with low-pass filtering, the orientation estimate will become good enough to do docking from 1m away. To continue this thesis, the author recommends to remove the EKF and add a low-pass filter. Next, try to remove vicon input and maintain a stable hover from the mock tailsitter at about 0.5m-1m distance. An impressive milestone that showcases the ability to localize and dock might be visual servoing, that is, having the Hummingbird follow the mock tailsitter as somebody holds it infront of the camera and moves it.

References

[1] Jack Stewart, "Google tests drone deliveries in Project Wing trials - BBC News," 2014.

[2] A. Barr and T. Greenwald, "Google Working on New Drone After 'Wing' Design Failed - Digits - WSJ," 2015.

[3] P. Abouzakhm and I. Sharf, "Guidance, Navigation, and Control for Docking of Two Cubic Blimps," *IFAC-PapersOnLine*, 2016.

[4] R. Oung, A. Ramezani, and R. D'Andrea, "Feasibility of a distributed flight array," in *Proceedings of the IEEE Conference on Decision and Control*, 2009.

[5] R. Oung and R. D'Andrea, "The distributed flight array," *Mechatronics*, 2011.

[6] Y. Wu, *Design and Implementation of an Unmanned Dual-Rotor Tail-sitter Vehicle: First Steps Towards UAV Autonomous Airborne Docking*. Bachelor thesis, University of Toronto, 2018.

[7] L. Meier, D. Honegger, and M. Pollefeys, "PX4: A node-based multithreaded open source robotics framework for deeply embedded platforms," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 6235–6240, IEEE, may 2015.

[8] V. Ermakov, "mavros." Available at http://wiki.ros.org/mavros.

[9] M. Fiala, "ARTag, a fiducial marker system using digital techniques," in *Proceedings - 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, CVPR 2005*, 2005.

[10] E. Olson, "AprilTag: A robust and flexible visual fiducial system," in *Proceedings - IEEE International Conference on Robotics and Automation*, 2011.

[11] J. Wang and E. Olson, "AprilTag 2: Efficient and robust fiducial detection," in *IEEE International Conference on Intelligent Robots and Systems*, 2016.

[12] B. Atcheson, F. Heide, and W. Heidrich, "Caltag: High precision fiducial markers for camera calibration," in *Vision Modeling and Visualization - VMV*, 2010.

[13] A. Sagitov, K. Shabalina, L. Sabirova, H. Li, and E. Magid, "ARTag, AprilTag and CALTag Fiducial Marker Systems: Comparison in a Presence of Partial Marker Occlusion and Rotation," 2017.

[14] D. Malyuta, "Guidance, Navigation, Control and Mission Logic for Quadrotor Full-cycle Autonomy," master thesis, Jet Propulsion Laboratory, 4800 Oak Grove Drive, Pasadena, CA 91109, USA, Dec. 2017.

[15] D. Malyuta, "apriltags2_ros," 2016. Available at http://wiki.ros.org/apriltags2.

[16] T. D. Barfoot, *State estimation for robotics*. 2017.

[17] A. Almagbile, J. Wang, and W. Ding, "Evaluating the Performances of Adaptive Kalman Filter Methods in GPS/INS Integration," tech. rep.

[18] M. Herb, "kalman." Available at: https://github.com/mherb/kalman.

[19] "MATLAB ROS interface." Available at https://www.mathworks.com/help/robotics/examples/get-started-with-ros.html.

[20] "QGroundControl." Available at: http://qgroundcontrol.com/.

[21] J. Kelly and G. S. Sukhatme, "Visual-inertial sensor fusion: Localization, mapping and sensor-to-sensor Self-calibration," *International Journal of Robotics Research*, 2011.

A Noise Covariance Matrices

State noise covariance matrices:

$$\Sigma_o = \begin{bmatrix} 0.1 & 0 & 0 \\ 0 & 0.1 & 0 \\ 0 & 0 & 0.1 \end{bmatrix}, \Sigma_\eta = \begin{bmatrix} 0.01 & 0 & 0 \\ 0 & 0.01 & 0 \\ 0 & 0 & 0.01 \end{bmatrix}$$

$$\Sigma_v = \begin{bmatrix} 0.1 & 0 & 0 \\ 0 & 0.1 & 0 \\ 0 & 0 & 0.1 \end{bmatrix}, \Sigma_{aw} = \begin{bmatrix} 0.01 & 0 & 0 \\ 0 & 0.01 & 0 \\ 0 & 0 & 0.01 \end{bmatrix},$$

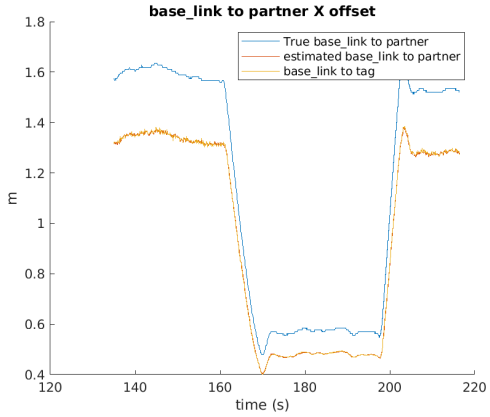
Accelerometer/Gyroscope input covariance matrices (multiplied by dt):

$$\Sigma_\omega = \begin{bmatrix} 0.001 & 0 & 0 \\ 0 & 0.001 & 0 \\ 0 & 0 & 0.001 \end{bmatrix}, \Sigma_a = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 2 \end{bmatrix}$$

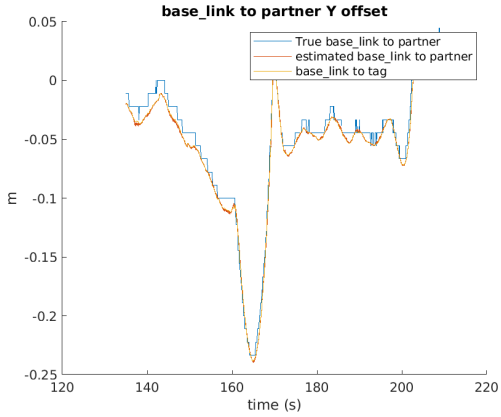
Measurement covariance matrices:

$$\Sigma_{\eta m} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \Sigma_{om} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

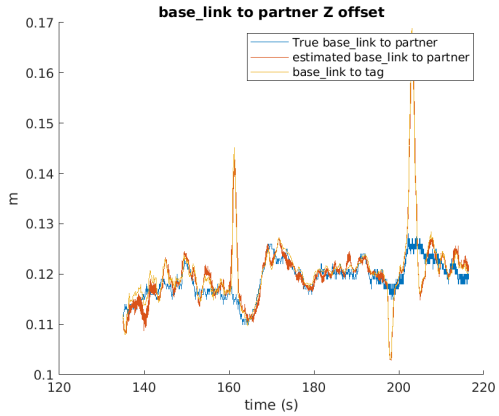
B Pose Estimate: Simulated Approach Trajectory



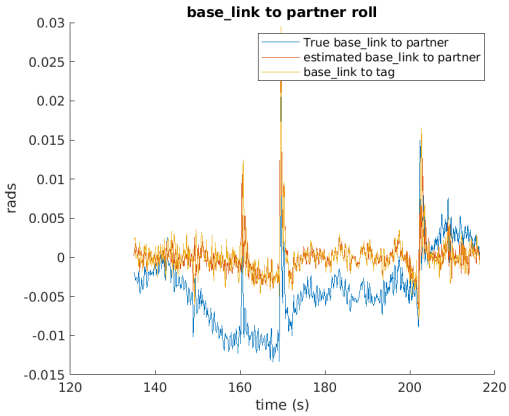
(a) Target tailsitter x estimate.



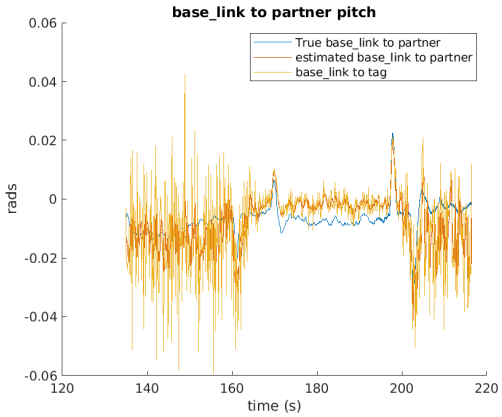
(b) Target tailsitter y estimate.



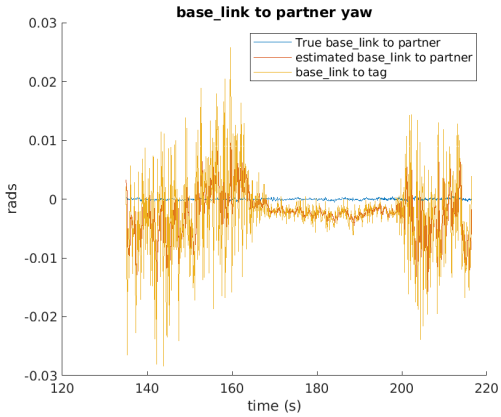
(c) Target tailsitter z estimate.



(d) Target to base_link roll estimate.

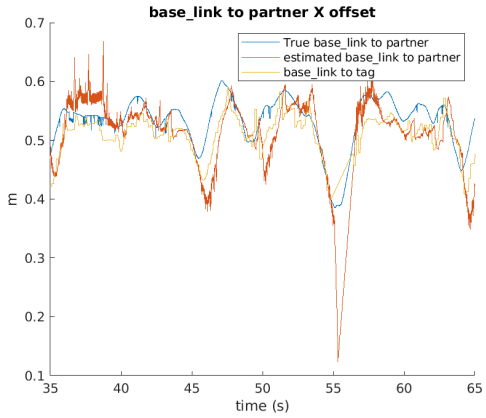


(e) Target to base_link pitch estimate.

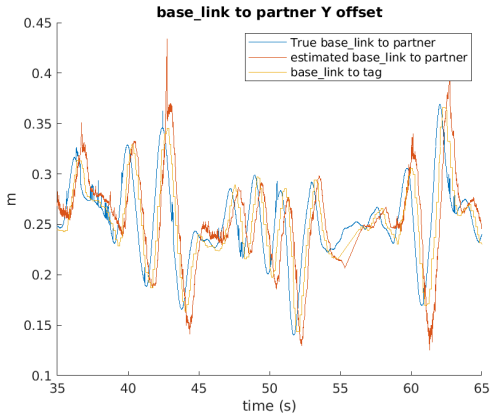


(f) Target to base_link yaw estimate.

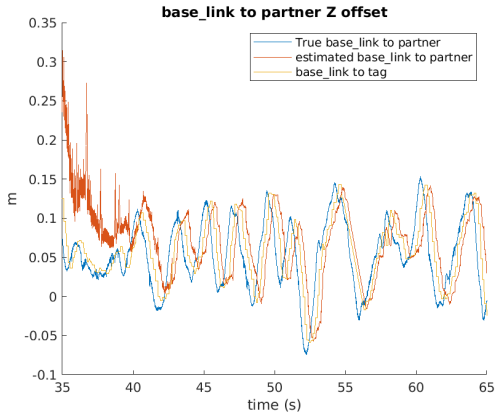
C Pose Estimate: 0.5m Hover



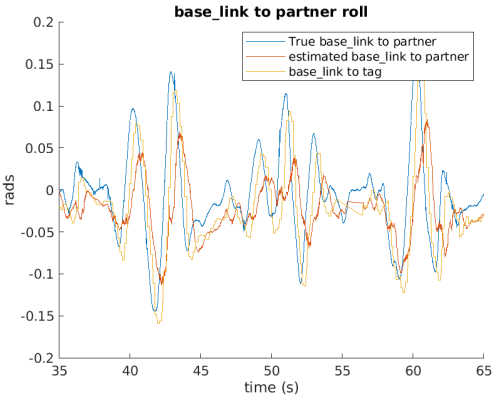
(a) Target tailsitter x estimate.



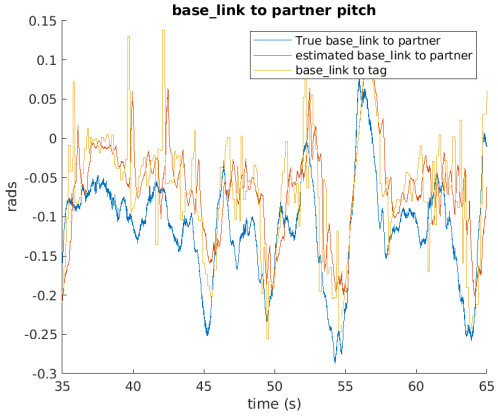
(b) Target tailsitter y estimate.



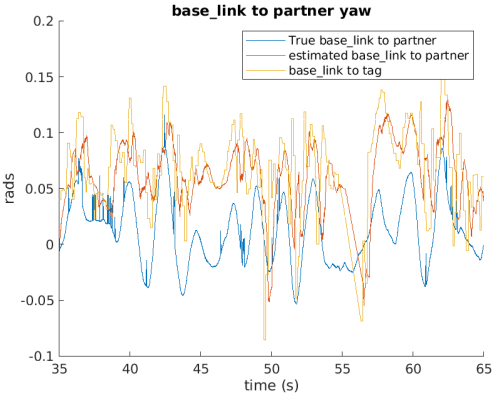
(c) Target tailsitter z estimate.



(d) Target to base_link roll estimate.

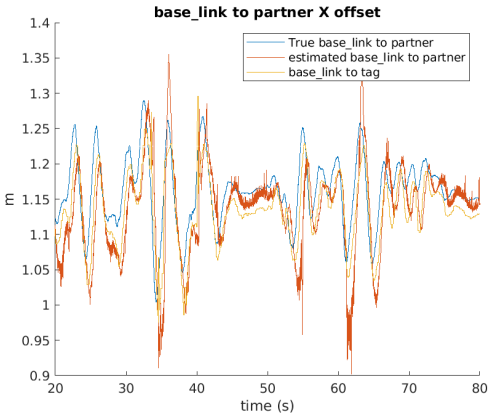


(e) Target to base_link pitch estimate.

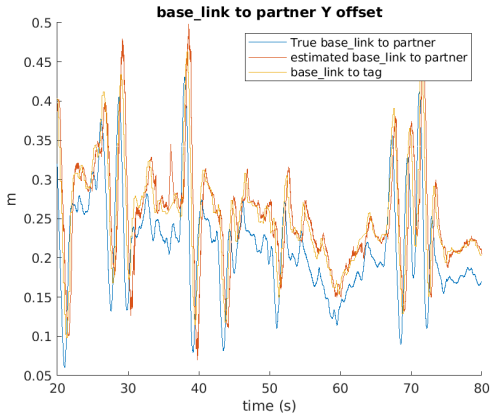


(f) Target to base_link yaw estimate.

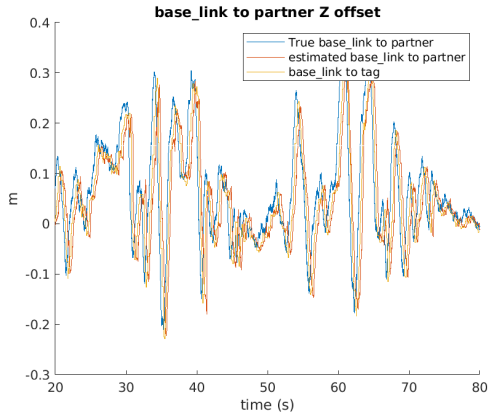
D Pose Estimate: 1m Hover



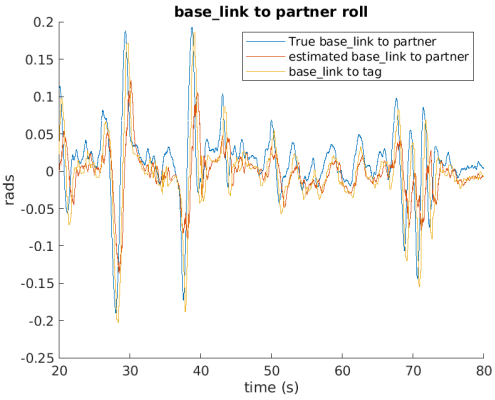
(a) Target tailsitter x estimate.



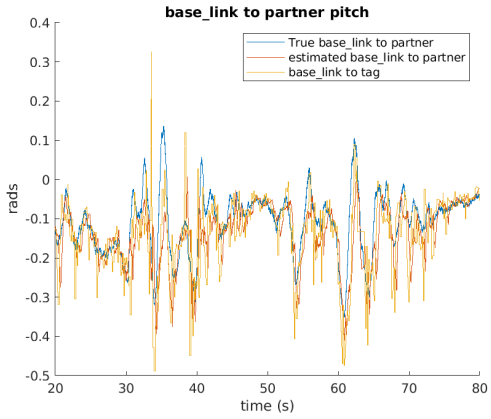
(b) Target tailsitter y estimate.



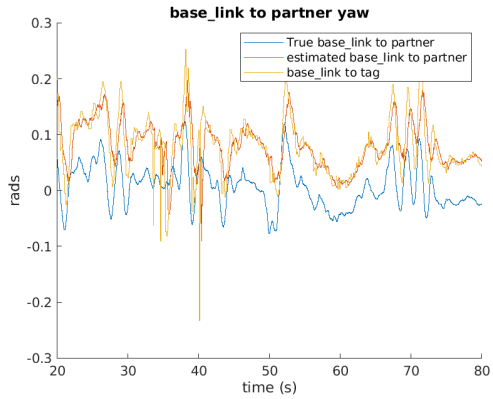
(c) Target tailsitter z estimate.



(d) Target to base_link roll estimate.



(e) Target to base_link pitch estimate.



(f) Target to base_link yaw estimate.

This page was intentionally left blank.