Autonomous Quadcopter with Delta Arm Manipulator Based on Motion Capture System

Zihang Wei
Electrical Computer Engineering
University of Michigan
Ann Arbor, Michigan 48105
Email: wzih@umich.edu

Prashin Santosh Sharma
Aerospace Engineering
University of Michigan
Ann Arbor, Michigan 48105
Email: prashinr@umich.edu

Eric Tsai
Robotics Institute
University of Michigan
Ann Arbor, Michigan 48105
Email: ericct@umich.edu

The Great Jerry
Robotics Institute
University of Michigan

***Abstract*—Quadcopters are an increasingly popular technology, emerging in applications that include package delivery services and traversing human-inaccessible environments. Met with both excitement and controversy, autonomous control of quadcopters is a topic essential to the technology's success in the future. In this project, we implement an autonomous control system for a quadrotor with the ultimate goal to transit to a particular location, pick up a small block, and return it to a destination point and drop the block. To achieve this, the project involves implementation of PID controllers, state machines, signal processing of Optitrack data, mechanical design and 3D-printing skills, as well as delta arm kinematics.**

Keywords - Quadcopter, Cascade control, Path planning, State machine, LCM, Delta arm, gripper

I. Introduction

The objective of the project is to implement an autonomous control system for a quadcopter using the Optitrack camera system as feedback for the state of the robot. In addition, a delta arm is equipped to the quadrotor along with a custom-made end-effector for picking up objects. Ultimately, the quadrotor should be able to fly to a commanded waypoint to pick up a block and place it at another destination. The project involves the integration of forward and inverse kinematics to control the delta arm. A 3D-printed gripper as an end-effector incorporates unique features, including an actuated bottom support, and an infrared sensor to determine that a block has been grasped. In addition, multiple Proportional-Integral-Derivative controllers are used to stabilize the system when hovering in place or traveling between waypoints. Finally, state machines are integrated to determine the quadcopter's progress in completing a sequence of tasks. By incorporating these components into a single functional system, we successfully demonstrate the quadrotor's ability to autonomously execute block pick-and-place tasks.

II. System Overview

In this section, we discuss the mechanical and electrical composition of the quadrotor. This includes the hardware that facilitate communication and autonomous control capabilities, as well as the delta arm/gripper assembly that executes block pick-and-place actions.

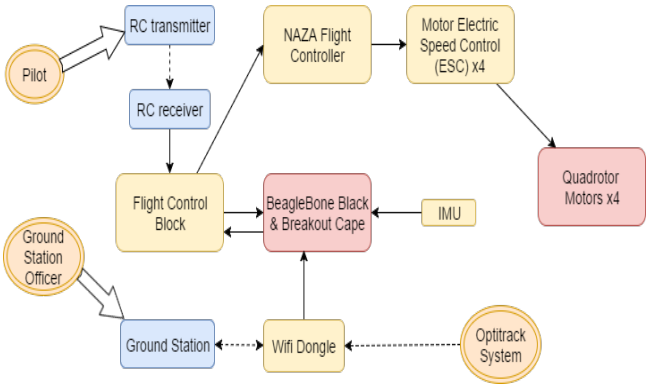

Fig. 1: Robot electrical design

A. Electric Structure

Our robot system includes several electrical modules that interface with one another, as shown in Fig.1. The subsystems are listed below.

`BeagleBone Black(BBB)`: The BeagleBone is a low-power, open-source, single-board computer running Debian OS. In this project, because of certain kernel patch problems, we can only use it as a near real-time control system. This system proves sufficient for the implementation of all desired functionalities like the cascade control scheme and data acquisition through IMU interrupts,etc.

`Breakout Cape`: The Breakout Cape is an extension and interface of the BeagleBone board. It has screw terminals for power, twelve GPIO pins, seven ADC channels, two UARTs, one I2C bus (I2C2), one SPI bus (SPI0) and a second SPI bus that can be run in real time from the programmable real-time units (PRUs) and five 3-pin headers for hobby servos. This is used to primarily isolate and protect the BBB from potentially damaging back EMFs produced by the motors.

`IMU`: The MPU-9250 (IMU) contains an accelerometer, gyro and magnetometer. In addition, it has a special digital motion processor that can output fused orientation data in the form of quaternions and Tait-Bryan (Euler) angles.

`NAZA PMU`: The DJI Naza M PMU V2 is a compact power management unit (PMU) that features enhanced BEC functionality and extendable CAN BUS ports.

`NAZA Controller` : Naza M V2 is a lightweight all-in-one multi-axis control platform with independent PMU and function extension.
`Arduino Uno`: The Arduino Uno is an open source, real-time microprocessor board from which time-accurate data from an infrared sensor can be obtained to operate the gripper.
`Wifi Dongle`: We use the Wifi module to establish a communication channel between the laptop and BeagleBone.
`ESC module`: The electronic speed control module is an electronic circuit used to vary an electric motor's speed and direction. The ESC is used to control propeller motors.
`Blocks`: The blocks system allows intercepting and retransmitting signals sent from the radio receiver to the flight controller. This allows us to implement an outer-loop controller with the BeagleBone acting as a full or partial autopilot.
`Potentiometers`: A three-potentiometer module was assembled to simulate the trajectories of both the quadrotor and the delta arm. This system provided an alternative testbed for our state machine and arm kinematics without relying on Optitrack. The voltage inputs from the potentiometers, read by the BeagleBone ADC module, represented the xyz-coordinates of the quadrotor with respect to the world frame. Similarly, the microcontroller could be programmed to interpret the inputs as coordinates of the delta arm. The potentiometer system streamlined the testing process, allowing us to tune our programs' parameters without the high-demand Optitrack room. Details of this simulation tool are discussed in a later section.

B. Mechanical Structure

The quadrotor frame is comprised of carbon fiber, a material that is both lightweight and durable for high-pressure environments. Fitted below the frame is a delta arm, a manipulator ideal for high-speed applications. A custom-made gripper is attached as the end-effector and includes unique features to enhance block-grasping capabilities.

1) Gripper Design: The two-finger, rotation-based gripper utilizes a single XL-320 Dynamixel motor (Fig. 2). The fingers measure approximately 11.8 cm in width, enough to adequately hold two blocks at once. This compensates for turbulence experienced by the quadcopter during the pick-up phase, increasing the likelihood of grasping a block.

The design also integrates housing for a Tower Pro micro servo, which actuates a bottom support after grasping a block. This acts as contingency plan in the event that the gripper cannot secure the block after pick-up.

In addition, a slot is provided for an IR sensor on the side of the gripper. The sensor is used to confirm that a block has been picked up. The gripper is equipped with an adjustable rod, which is passed between the beacons of the infrared (IR) sensor, indicating that a block has been picked up. Both the IR sensor and servo motor are controlled by an Arduino Uno microcontroller, fitted onto the landing gear of the quadrotor.

During testing, the Arduino was connected to the USB hub used by the BeagleBone. While this was a convenient setup, the Arduino was effectively being powered by the BeagleBone, which could not function properly. In order to fully implement our gripper design, we would need to introduce a separate power supply (another battery pack) on the quadrotor. However, with limited time and concerns of over-encumbering the quadcopter, the gripper was simplified by omitting the IR sensor and bottom-support features. Hand-carried delta arm tests demonstrated that the gripper could consistently pick up blocks even without this equipment. In the end, Jerry did not get to play with his new toys for the competition.

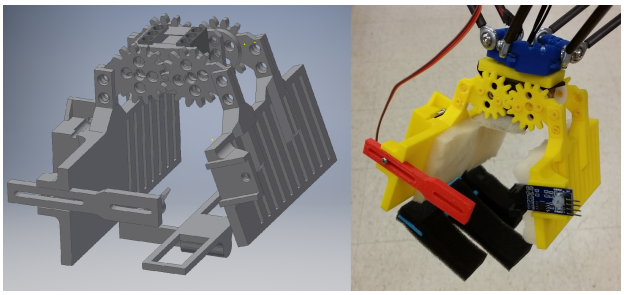

Fig. 2: CAD design and 3D-printed gripper

2) Delta Arm Manipulator: The quadcopter is equipped with a delta arm and gripper in order to grasp blocks. The arm consists of three struts, each actuated by an XL-320 Dynamixel motor. The dimensions of the arm are outlined in Table I and Figure 3. The struts are connected between the base of the quadrotor and the end effector. With the struts configured to form planar parallelograms, the end effector platform remains parallel to the quadrotor base. Forward and inverse kinematics are implemented using pre-existing code provided by a Delta robot kinematics tutorial from Trossen Robotics Community.

a) Forward Kinematics: Given the motor's angles θ_1, θ_2, and θ_3, we can compute the end effector's coordinates E_0. The joints J_1E_1, J_2E_2, and J_3E_3 freely rotate within a sphere of radius r_f with centers at J_1, J_2, and J_3 (Eq. 1). By translating the coordinates of these joints to J_1', J_2', and J_3', the joints' respective spheres intersect at the single point of E_0. This relationship is defined in a system of equations (Eq. 2 and 3) to solve for E_0 (Eq. 4).

$$\begin{aligned} J_1' &= (0, -(t + r_f cos(\theta_1)), -r_f sin(theta1)) \\ J_2' &= (y_2 tan(60°), (t + r_f cos(\theta_2))sin(30°), -r_f sin(\theta_2)) \\ J_3' &= (-y_3 tan(60°), (t + r_f cos(\theta_3))sin(30°), -r_f sin(\theta_3)) \end{aligned} \tag{1}$$

$$\begin{aligned} a &= a_1^2 + a_2^2 + d_{nm}^2 \\ b &= 2(a_1 b_1 + a2(b_2 - y_1 d_{nm}) - z_1 d_{nm}^2) \\ c &= (b_2 - y_1 d_{nm})(b_2 - y_1 d_{nm}) + b_1^2 + d_{nm}^2(z_1^2 - r_e^2) \end{aligned} \tag{2}$$

where

$$
\begin{aligned}
a_1 &= (z_2 - z_1)(y_3 - y_1) - (z_3 - z_1)(y_2 - y_1) \\
b_1 &= -\frac{(w_2 - w_1)(y_3 - y_1) - (w_3 - w_1)(y_2 - y_1)}{2} \\
a_2 &= -(z_2 - z_1)x_3 + (z_3 - z_1)x_2 \\
b_2 &= \frac{((w_2 - w_1)x_3 - (w_3 - w_1)x_2)}{2} \\
d_n m &= (y_2 - y_1)x_3 - (y_3 - y_1)x_2 \\
w_i &= x_i^2 + y_i^2 + z_i^2 \text{ for } i = 1, 2, 3
\end{aligned} \tag{3}
$$

$$
\begin{aligned}
z_0 &= -\frac{0.5(b + \sqrt{d})}{a} \\
x_0 &= \frac{a_1 z_0 + b_1}{d_{nm}} \\
y_0 &= \frac{a_2 z_0 + b_2}{d_{nm}}
\end{aligned} \tag{4}
$$

where discriminant $d = b^2 - 4.0ac$ must satisfy $d < 0$ for a valid joint angle configuration.

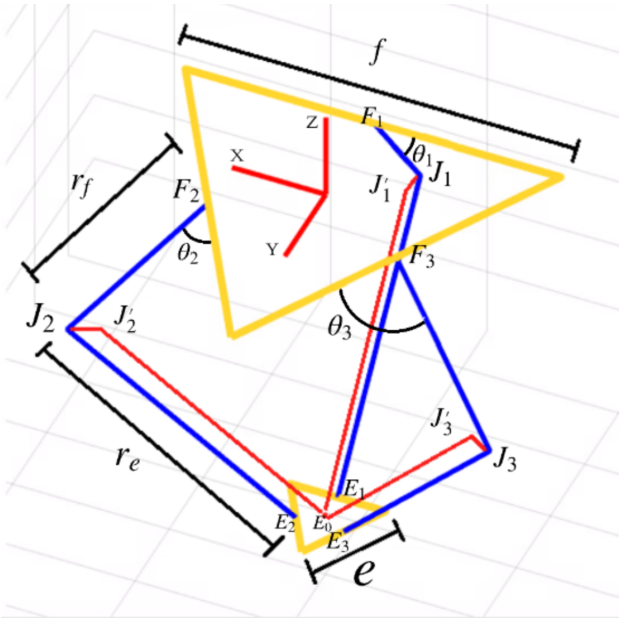

Fig. 3: Diagram of delta arm

Component	Description	Length (cm)
r_f	Top link length	11.9
r_e	Bottom link length	19.5
f	Base plate side length	24.0
e	Bottom plate side length	6.1

TABLE I: Delta arm dimensions

b) Inverse Kinematics: Inverse kinematics were implemented in order to command the end effector to a specific position. As seen in Figure x, the joint J_1F_1 is constrained to the YZ-plane, with a circular range of motion of radius r_f. The joint J_1E_1, however, is a universal joint that has a spherical range of motion of radius r_e. The intersection of this sphere and circle from J_1F_1 determines the position of joint J_1, and thus servo position θ_1 (Eq. 5 and 6). The same procedure can be used to calculate angles θ_2 and θ_3, whereby the XY-plane is rotated counter-clockwise by 120° to align with the next joint (Eq. 7 and 8).

$$
\theta = \begin{cases} \frac{180}{\pi} \arctan \frac{-z_j}{y_1 - y_j} + 180, & \text{if } y_j > y_1 \\ \frac{180}{\pi} \arctan \frac{-z_j}{y_1 - y_j}, & \text{otherwise} \end{cases} \tag{5}
$$

where

$$
\begin{aligned}
y_1 &= -\frac{f}{2} tan(30°) \\
y_0 - &= \frac{e}{2} tan(30°); \\
a &= (x_0^2 + y_0^2 + z_0^2 + r_f^2 - r_e^2 - y_1^2)/(2z_0) \\
b &= \frac{(y1 - y0)}{z0} \\
d &= -(a + by_1)^2 + r_f(b^2 r_f + r_f) \\
y_j &= \frac{y_1 - ab - \sqrt{d}}{b^2 + 1} \\
z_j &= a + by_j
\end{aligned} \tag{6}
$$

where discriminant $d < 0$ in order to satisfy a valid joint angle configuration.

$$
\begin{aligned}
x_0^{(3)} &= x_0 cos(120°) + y_0 sin(120°) \\
y_0^{(3)} &= y_0 cos(120°) - x_0 sin(120°) \\
z_0^{(3)} &= z_0
\end{aligned} \tag{7}
$$

$$
\begin{aligned}
x_0^{(2)} &= x_0 cos(120°) - y_0 sin(120°) \\
y_0^{(2)} &= y_0 cos(120°) + x_0 sin(120°) \\
z_0^{(2)} &= z_0
\end{aligned} \tag{8}
$$

III. Software Design

In this section, we outline the software architecture and state machine that serve as an intuitive and systematic procedure for pick-and-place tasks.

A. Software Architecture

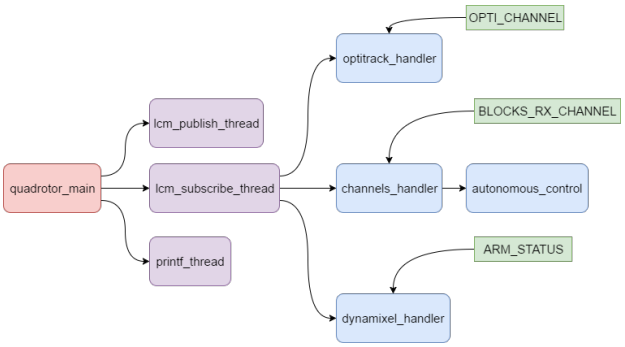

Fig. 4: Software Architecture

As shown in Fig. 4, a multi-thread software system is implemented for control of the quadrotor. The main function initializes all configurations and threads, which carry out tasks that include printing the quadrotor's state to the command line (`printf_thread`) and recording LCM logger data (`lcm_publish_thread`). In addition, `lcm_subscribe_thread` is dedicated to subscribing to appropriate LCM channels to acquire information, such as Optitrack, RC transmitter, and Dynamixel motor data.

When LCM messages are received, corresponding handler functions are called to appropriately process the information. The `channels_handler`, `optitrack_handler`, and `dynamixel_handler` functions are triggered by the `BLOCKS_RX_CHANNEL`, `OPTI_CHANNEL`, and `ARM_STATUS` channels respectively. In `channels_handler`, if the auto mode is switched on (RC transmitter $channel[7] > 1500$), the quadrotor enters into autonomous mode and runs the `autonomous_control` function. All of the PID controllers and the state machine are implemented in the `autonomous_control` function, which updates the new channel commands and publishes them to the NAZA controller.

B. Potentiometer-based Simulation System

One of the challenges during the project was sharing access to the Optitrack room with other teams for flight tests. To streamline the testing process, we devised a simulation tool that avoids the need for Optitrack.

We used a three-potentiometer module to simulate the trajectories of both the quadrotor and the delta arm. We connected the BeagleBone with a block and the RC receiver to trigger the `channels_handler` function, and used the Optitrack signal to trigger `optitrack_handler` function. The voltage inputs from the potentiometers, read by the BeagleBone ADC module, represented the xyz-coordinates of the quadrotor with respect to the world frame. In the `optitrack_handler` function, we populated the `pose[i]` and `set_pose[i]` variables with our simulated values from the potentiometers or prefixed data. We could also change our program to interpret the potentiometer input as coordinates of the delta arm to test the orientation and accuracy of its inverse kinematics.

Regarding the stability of the quadrotor, the PID tuning process still required Optitrack feeback. However, integration of the simulation tool provided flexibility, allowing us to continue testing the quadrotor even when Optitrack was being utilized by other teams.

C. State Machine and Decision Making Architecture

The state machine is used by the program to determine the next step that needs to be taken by the robot. The state of the robot is continuously checked before setting the next state.

For a pick-and-place sequence we include following steps: (1) Pick-and-place waypoints are set based on the coordinates of the pick-up and drop-off platforms acquired from Optitrack. (2) The pilot manually takes off, then initiates

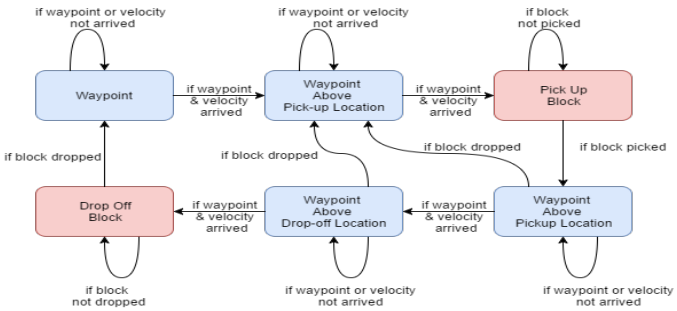

Fig. 5: State machine for pick and drop task

autonomous control for the BeagleBone. (3) The quadrotor transits to the pick-up site, and (4) the delta arm/gripper picks up the block. (5) Next, the quadrotor heads to the drop-off receptacle for block release, and (6) the gripper releases the block at the drop-off location. (7) Finally, the quadrotor returns to its initial position. The sequence is then repeated to place multiple blocks in the receptacle.

`waypoint and velocity check`: Ensure that the current pose reaches the target location and the quadrotor's velocity remains within a certain error range.

`grab and drop check`: Only when the gripper's angle position and torque are in specified ranges can we consider the block grabbed or dropped. Between the pick-up state and drop-off state, if the block is detected as dropped, the quadrotor returns to the pick-up location to re-pick the block.

D. Gripper State Machine

A simple state machine is implemented to actuate the bottom support of the gripper when a block has been grasped (Fig. 6), which is indicated by the IR sensor. A rod fitted to one side of the gripper contains a protruding tab (about 1 cm long), which passes between the beacons of the IR sensor when the fingers close. Successful block acquisition is confirmed when the tab remains within range of the sensor for an amount of time specified by the variable `grab_threshold`. An IR variable is set to 1 when the tab is detected by the IR sensor and is set to 0 otherwise. With a value of 1, the variable `grab_check_counts` is incremented continuously until it meets the threshold, at which point the bottom support is deployed. If the IR variable is 0 and `grab_check_counts` $<$ `grab_threshold`, then this is considered environment noise and `grab_check_counts` is reset. Such an event can occur if the gripper misses the block and the fingers fully close, causing the tab to overshoot and pass through the sensor too quickly. The `grab_threshold` variable also gives the quadrotor time after gripping the block to increase altitude so that the support does not collide with the pick-up platform. Similarly, when the block is being released at the designated drop-off point, the tab must be retracted from the IR sensor's range for the period defined by `drop_threshold`, at which time the support is retracted. The video *Gripper_state_machine* is provided to demon-

strate the state machine in action. As mentioned earlier, due to complications integrating the Arduino Uno into our design, this state machine was not utilized during the competition. Instead, we simplified our state machine by utilizing torque feedback and angle position of the gripper's servo motor.

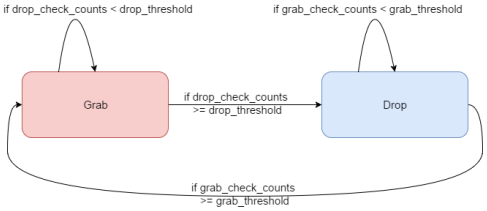

Fig. 6: State machine for gripper's bottom support deployment

IV. Modelling of Quadrotor

The quadrotor is one of the simplest multi-rotor vehicles. The SkySpecs quadrotor we use employs differential control of thrust generated by each of the rotors as shown in Fig. 7. As the quadrotor is flown indoors we ignore the forces due to aerodynamic effects.

The steady-state thrust generated by the quadrotor can be modeled as the following:

$$T_i = C_T \rho A_{r_i} r_i^2 \omega_i^2 \tag{9}$$

where for rotor i, A_i is the rotor disk area, r_i is the radius, ω_i is the angular velocity, C_T is the thrust coefficient and ρ is the density of air. However, for our model we will be using only a lumped parameter model given as:

$$T_i = C_T \omega_i^2 \tag{10}$$

The reaction torque due to the rotors is modeled as the following:

$$Q_i = C_Q \omega_i^2 \tag{11}$$

where C_Q can be determined by a stationary thrust test. Hence, the total thrust and the net moment can be combined into matrix form and written as:

$$\begin{pmatrix} T_\sigma \\ \tau_x \\ \tau_y \\ \tau_z \end{pmatrix} = \begin{pmatrix} C_T & C_T & C_T & C_T \\ ds\theta C_T & -ds\theta C_T & ds\theta C_T & -ds\theta C_T \\ dc\theta C_T & dc\theta C_T & -dc\theta C_T & -dc\theta C_T \\ -C_\theta & C_\theta & -C_\theta & C_\theta \end{pmatrix} \begin{pmatrix} \omega_1^2 \\ \omega_2^2 \\ \omega_3^2 \\ \omega_4^2 \end{pmatrix} \tag{12}$$

where s_θ and c_θ are shorthands for Sine and Cosine respectively.

V. Control System Design

We use the BeagleBone Black (BBB) for position and velocity control, as well as the NAZA controller to convert the BBB commands into differential thrust control outputs for each rotor. Initially a remote control is used for taking off, and once the quadrotor is stabilized, autonomy mode is activated. Then, the BBB takes control, transmitting the desired PWM signals to the NAZA based on the task at hand and the feedback from Optitrack.

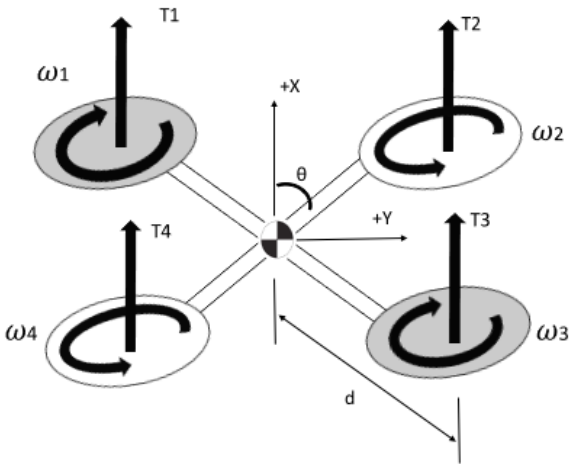

Fig. 7: Force diagram for quadrotor

For the control system, a two-layer scheme is implemented as shown in Fig.8. The first layer is a distance interpreter, which computes a desired velocity for the quadrotor based on the distance between its current position and waypoint. The second layer is a trajectory controller, which ensures that the quadrotor follows the trajectory set by the state machine while moving at the desired velocity. Output of the second layer serves as input to the NAZA's inner loop controller, which ultimately drives the quadrotor motors.

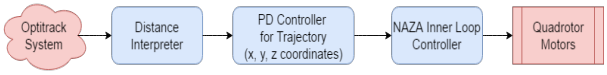

Fig. 8: Control block logic

A. Cascaded Control

The two-layer control scheme can be viewed as a cascaded controller, where `distance interpreter` is a saturated P controller as shown in Fig. 9. We first build the theoretical model and estimate the slope rate and cut-off value, then evaluate these parameters during flight tests. Our parameters for this `distance interpreter` can be seen from Fig. 9.

The PID controller is a control loop feedback system that is commonly used in industry. It continuously calculates an error value $e(t)$ as the difference between a desired set-point and a measured process variable. Then, it applies a correction based on proportional, integral, and derivative gains. The output of the controller is the weighted sum of these three terms:

$$u(t) = K_p e(t) + K_i \int_0^t e(t)dt + K_d \frac{de(t)}{dt} \tag{13}$$

In our control system, we only use a PD controller since they are easy to tune in a short amount of time. Also, we do not

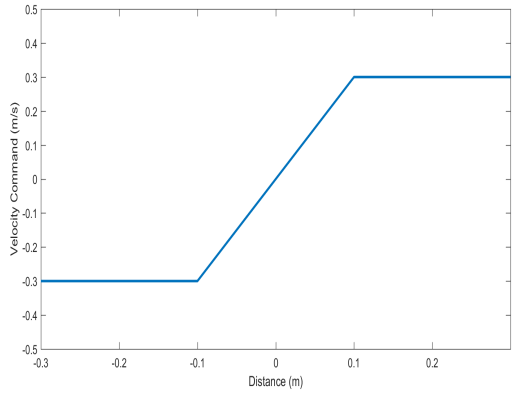

Fig. 9: Quadcopter velocity based on distance from waypoint

fly the quadrotor close to the floor, which would cause wind disturbance and thus substantial drift error, so an integral term is not crucial. Following the Ziegler-Nichols tuning method, we first set I(integral) and D(derivative) gains to zero. K_p is then increased until it reaches the gain K_u at which the output of the control loop has stable and consistent oscillations in hovering or transient mode.

The `trajectory controller` model consists of four PD controllers to control coordinates x, y, z and angle yaw. Through experimentation, we found that the yaw angle is quite stable and does not drift. Therefore, the yaw controller was omitted for the final competition.

In this project for the PD controller parameters as shown in Table II

TABLE II: PID Gains

	X	Y	Z
K_p	200	200	200
K_d	50	50	50

B. Filter for Optitrack Data

Directly processing the pose and velocity data from the Optitrack often introduces unnecessary noise into our system, resulting in random oscillations of the quadrotor. The noisy data is more noticeable when fewer cameras are used for position tracking or the quadcopter is out of range for a subset of the cameras. To acquire more stable readings, we apply a low-pass filter to the Optitrack data stream, which reduced unexpected movements in the quadcopter during flight.

VI. Task Design and Performance

In order to evaluate the performance of the quadrotor/delta arm design, we use several test cases to fly the quadrotor in autonomous mode. During such flight tests, the robot is attached to two tethers as a safety precaution.

A. Waypoint Hold

The first task of the competition involves the quadrotor hovering at a single position for an appreciable period of time. During the competition, the robot was subjected to disturbances, which included pulling it by its safety tethers in various directions (left, right, up, down, forward and backward). Our outer-loop control scheme proved effective, with minimal signs of drift or oscillations while hovering in place. In addition, the quadcopter recovered quickly when pulled in any direction and maintained its position as shown in Fig. 10.

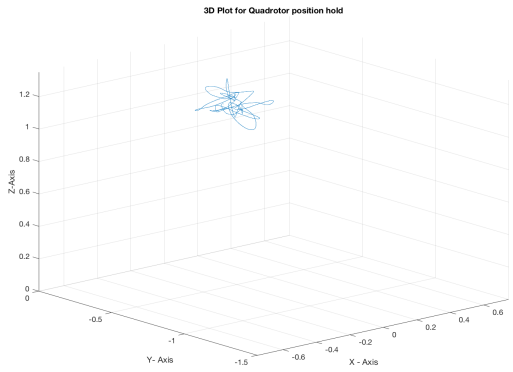

Fig. 10: 3D Plot of Quadrotor Center for waypoint hold with external disturbances

B. Waypoint Transit

The quadrotor is also programmed to traverse to one waypoint, hover briefly, then return to its initial position. For this task, the waypoint was set at the xy-coordinate (1.5 m,-1.5 m) with respect to the origin defined in the Optitrack space. This position was near the boundary of the Optitrack's range of vision. Therefore, it was essential that the quadrotor not overshoot its destination, which would result in loss of feedback. Two options for addressing this issue are tuning PID gains for the outer-loop controller or reducing the velocity of the robot when traveling between waypoints. Due to the time constraints of the competition, we decided to reduce the quadcopter's velocity, which resulted in successful completion of the task.

C. Block Pick-and-Place

Once waypoint hold and transit were successfully performed, the quadrotor was programmed to pick up a block from one location and drop it at another destination. As it hovers above the pick-up platform, it computes the distance of the block relative to the end-effector's position. When this distance is within a certain threshold, the delta arm is commanded to reach the block for pick-up.

The quadcopter was able to pick and place multiple blocks during the competition. In general our outer-loop controller allowed the robot to hover steadily above the pick-up platform while attempting to grasp the block. Fig. 11 demonstrates this

stability, showing the xyz-position of the robot over the course of the task. The changes in position are also relatively periodic (Fig. 11). The first two peaks do appear wider, meaning it took a longer time for the quadrotor to adjust its pose during both the pick and drop phases.

With the quadcopter lower to the ground during pick-up, more turbulence due to increased air pressure under the robot was experienced (known as the ground effect in aerodynamics). This can be gleaned from the small perturbations in the z-component in Fig. 11. Fortunately, the gripper compensated for this instability due to its wide fingers that provided a large surface area for catching the block.

Fig. 12 displays only the y-component of the robot for the entire duration of the pick-and-place task. We can see that each time the quadrotor reaches its destination, it overshoots slightly ($< 5\%$), resulting in a small spike at the edge of each peak in the plot. For the large spike observed on the final peak, the quadrotor overshot its destination and struggled to recover during the pick-up phase. This may be attributed to low battery power when the fifth block was attempted since the battery's indicator light was flashing rapidly at the time. It is also possible that the quadrotor briefly lost communication with the Optitrack. If the robot travels too far past the target and approaches the boundary of sight for the Optitrack, it may be delayed in stabilizing.

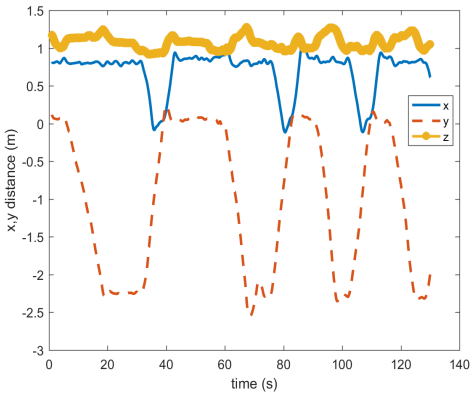

Fig. 11: Multi-rounds of time sequence plot for pick and drop task

Despite our overall success in this task, the gripper had a tendency to slightly overshoot when picking up the block, causing it to partially grip the block's platform. On the other hand, there were cases when the quadcopter was too high in altitude and could not reach the block. This is because we didn't measure the height offset in test area, and in order to avoid adjusting quadcopter pose for a very long time, we give the state machine a relatively large error check threshold (5cm). To solve this, we can refine the robot's decision of a valid pick-up distance by tuning offset and threshold parameters for the distance between the gripper and block.

As shown in Fig. 14, the pick location is roughly at (0.85 m, 0 m), the drop location at (0.8 m, -2.35 m), and we set

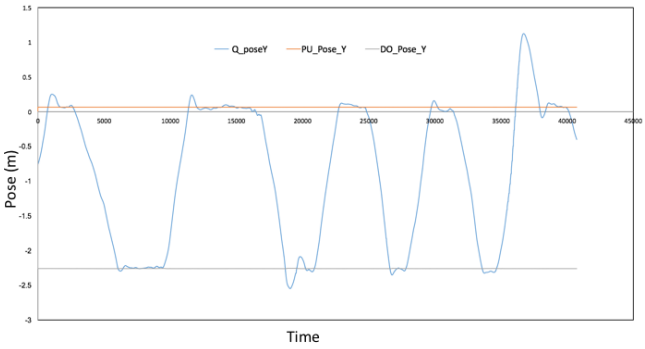

Fig. 12: 2D Plot of Center of quadrotor for pick place task with external disturbances

Fig. 13: Quadrotor pick and drop task

intermediate waypoints between the drop-off step and the next pick-up step. In Fig. 15, after four iterations of the task, the quadrotor overall shows high accuracy in following the same commanded trajectory. It can be seen, however, that the robot is particularly accurate when moving between the pick-up and drop-off stations. This is because in our design, we give a relatively small position error threshold for the quadcopter to follow between these two waypoints but allow larger thresholds (10cm 15cm) for the other waypoints. It is only crucial for the quadrotor to be exact in its position as it approaches the pick-up and drop-off locations.

D. Chasing a Mobile Drop-off Station

In this task, the goal is again to pick-and-place a block, but the drop-off station is now mobile and commanded remotely by a ground station operator. To account for the moving receptacle, we implement both location and velocity checks of the quadcopter. The location check verifies that the location error of the quadrotor and the current waypoint (the pick-up or drop-off station) are within a certain range. Furthermore, the velocity check ensures that the quadrotor matches the velocity of the drop-off station.

However, during the competition, we did not execute our velocity check function due to limited time in tuning velocity-

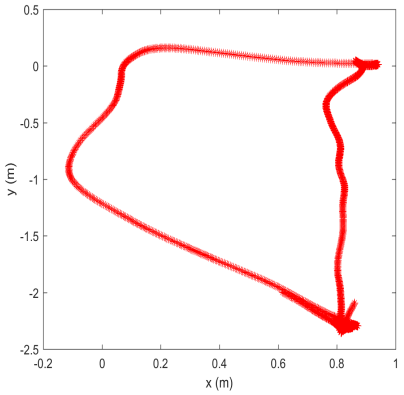

Fig. 14: One round of trajectory for pick and drop task

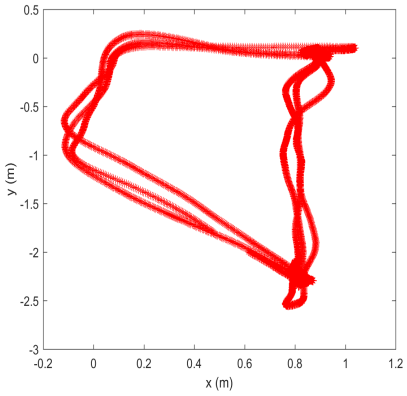

Fig. 15: Multi-rounds of trajectory for pick and drop task

check function parameters and error thresholds.

Despite relying only on the location check feature, the quadrotor was successfully able to drop one block into the mobile station, as also can be seen from Fig. 17. For the failed attempts, the quadrotor was in the process of releasing a block when the receptacle changed directions. Had the velocity check function been utilized, the quadcopter would have responded to the change and maintained its position above the station.

The `Chase a Mobile Drop-off Station` state machine differs from `Static pick-up and drop-off location` in that the delta arm is lowered down to close the distance between the block and the drop-off station. In previous trials, when the arm was not lowered, the dropped block took too long to reach the moving target. The block's trajectory could also be altered by air currents when dropped from a higher distance. Lowering the delta arm resulted in a more accurate drop-off performance during the competition. An alternative is to decrease the quadcopter's altitude, but this would only amplify the ground effect and cause more instability.

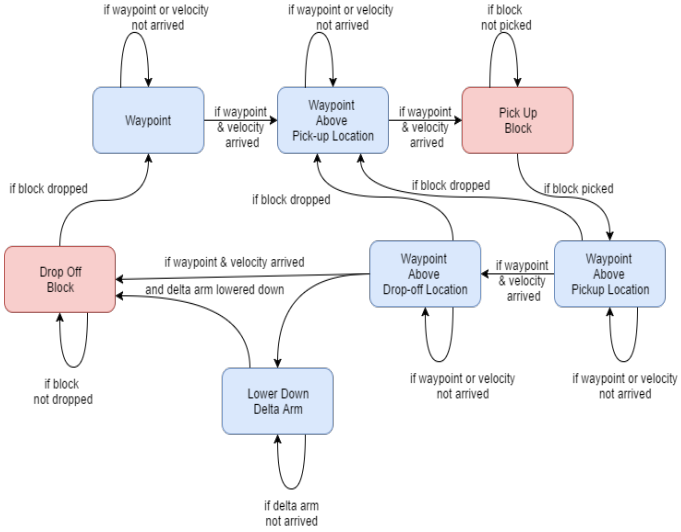

Fig. 16: State machine for pick and drop task when chasing mobile station

VII. Conclusion

The quadrotor successfully performed all tasks for the competition. The system's outer-loop controller proved robust to noise and perturbations while hovering in place. In addition, it demonstrated smooth transitions between waypoints. When picking and placing blocks, the delta arm/gripper assembly was effective. The extended width of the gripper's fingers was essential for the competition, increasing the chances of grasping the block. The state machine was sufficient for the tasks, allowing the quadrotor to travel between waypoints successfully, and execute gripper commands efficiently. An improvement of the state machine would be to integrate a velocity check for the pick-and-place task with a mobile station. By computing the velocity of the moving drop-off station, the quadrotor would have had a more accurate prediction as to where the station would be next, thus, allowing more blocks to be placed. In addition, incorporating the bottom-support and its state machine for the gripper would have further secured a grasped block. However, as the competition demonstrated, there were few instances in which the quadrotor picked up a block that slipped out before reaching the drop-out location. Therefore, the additional features of the gripper were not necessary, and may even have been more detrimental to our success due to the additional weight that would have existed with another servo motor, IR sensor, and support arm attached.

Although a yaw controller was not necessary for this project, it would have been helpful when grasping a block from the pick-up platform. There may be instances in which the gripper is oriented awkwardly with respect to the block and is forced to grasp the block by its corners rather than its sides. And with the delta arm unable to adjust its yaw effectively, implementing the yaw controller for the quadrotor would have improved our system's ability to manipulate blocks.

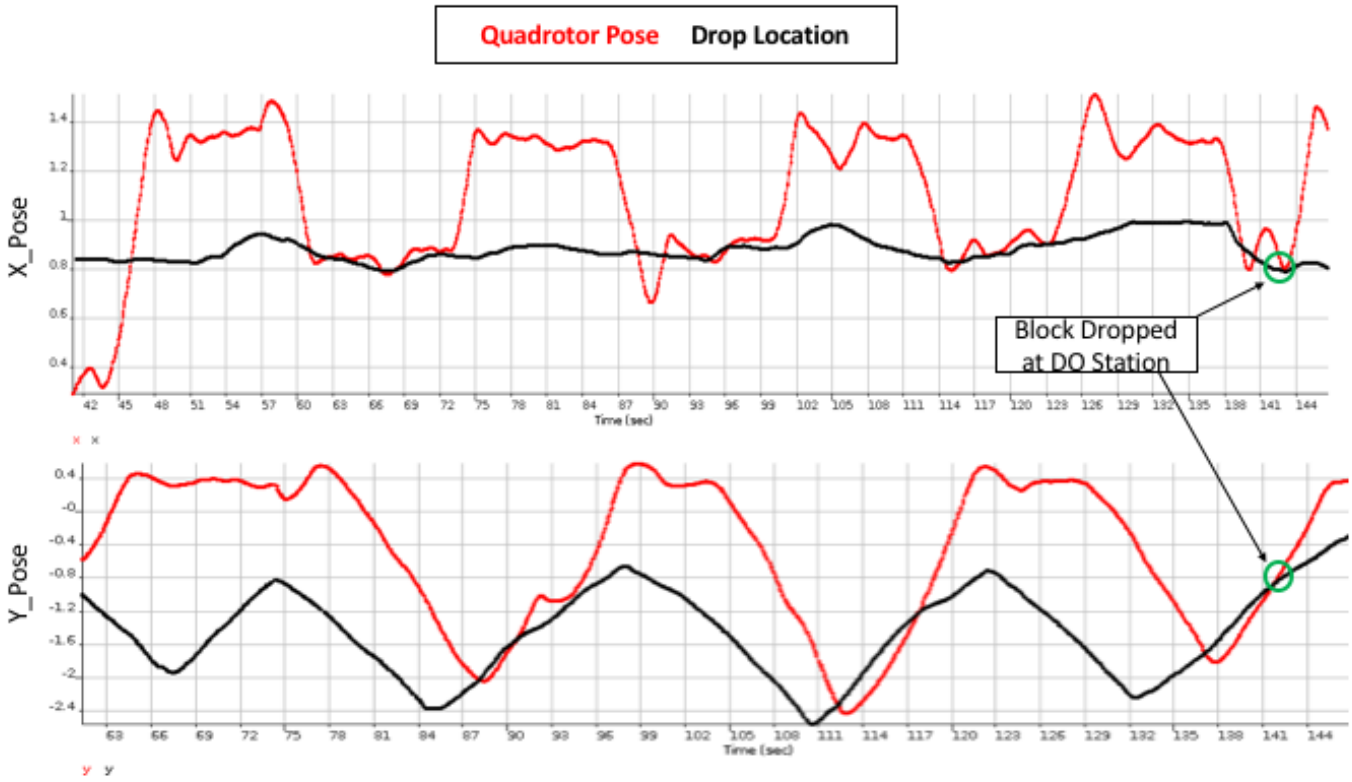

Fig. 17: X and Y Center of quadrotor plot against time while chasing the drop-off station

ACKNOWLEDGMENT

We would like to thank Prof. Ella Atkins, course instructor, for scientific guidance, Dr. Peter Gaskell, lab instructor, for providing us with the proper equipment set and guidance throughout the experimentation process and lab sessions, Mr. Theodore Nowak, graduate student instructor (GSI) and Mr. Abhiram Krishnan, instructional aide (IA) for comments, insight and expertise that greatly improved our understanding of all hardware and software systems. And of course, we thank the new-look Jerry for inspiring us every day to be better!

REFERENCES

[1] R. Mahony, V. Kumar and P. Corke, "Multirotor Aerial Vehicles: Modeling, Estimation, and Control of Quadrotor," in IEEE Robotics and Automation Magazine, vol. 19, no. 3, pp. 20-32, Sept. 2012. doi: 10.1109/MRA.2012.2206474

[2] "Delta robot kinematics" in Trossen Robotics Community, July 2009. http://forums.trossenrobotics.com/tutorials/introduction-129/delta-robot-kinematics-3276/. Accessed 5 April 2017.

[3] Spong, Mark W., Seth Hutchinson, and Mathukumalli Vidyasagar. Robot modeling and control. Vol. 3. New York: wiley, 2006.

VIII. Appendix

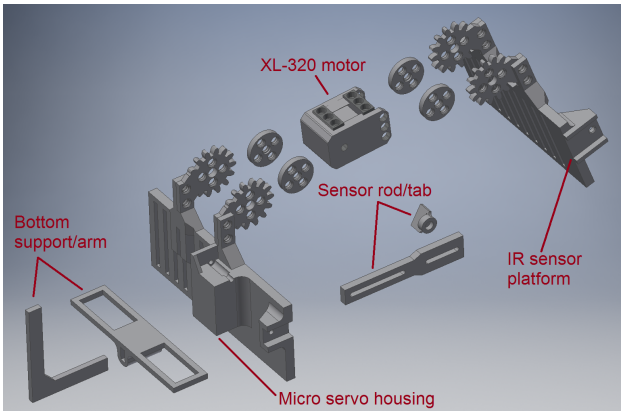

Fig. 18: Exploded view of gripper CAD design