Blocks Transport Quadrotor based on Optitrack Indoor Navigation System

Abhishek Venkataraman
Robotics Institute
University of Michigan
Ann Arbor, Michigan 48105
Email: wzih@umich.edu

Boliang Liu
Electrical Computer Engineering
University of Michigan
Ann Arbor, Michigan 48105
Email: boliang@umich.edu

Garrett Madsen
Department of Mechanical Engineering
University of Michigan
Ann Arbor, Michigan 48105
Email: gmadsen@umich.edu

***Abstract*—In this project we propose a methodology to implement a position outer loop controller for a quadrotor. The quadrotor has an existing well tuned inner loop controller. We also discuss the forward and inverse kinematics of a delta arm (payload/ manipulator). Finally, we propose a state machine to perform some predefined tasks and discuss its performance.**

***Index Terms*—quadrotor, delta-arm, grippers, state machines, PID controllers**

I. Introduction

Quadrotors have gained popularity in recent times for various purposes such as recreational, photography, real estate surveying, etc. Unlike fixed wing aircrafts, their ability to take off and land vertically(VTOL) has contributed to the wide array of applications that quadrotors now perform. The availability of low cost Inertial Measurement Units (IMU) and micro-controllers have made it possible it have a robust on board control system to autonomously control a quadrotor greatly increasing their usefulness.

In this project, we attempt to add an outer loop controller for reference Cartesian way points, that the quadrotor tracks using OptiTrack Motion Capture system. We further add the functionality to transit between way-points. Finally, we add a delta arm (with a gripper) on the under side of the quadrotor and use it to pick and drop blocks. With the development of a state machine, this combination of skills allow the quadrotor to autonomously fly to locations and pick up blocks and then proceed to fly to a drop off location. These abilities were formally tested in a series of competition tasks outlined in the report. Section VI-B discusses in detail our performance in these tasks.

II. Hardware Design

A. Quadrotor Dynamics

A quadrotor consists of four motors, two spinning clockwise and tw1o counter clockwise. Four motors is the minimum required configuration for changing pitch, roll, yaw and altitude. The position of the motors with respect to the quadrotor coordinate system is shown in Figure 1. The motion in each DoF is achieved by the following action:

- For a forward pitch, motors 4 and 3 generate more thrust than motors 1 and 2.
- For a positive roll, motors 1 and 4 generate more thrust than motors 2 and 3.
- For a CW yaw, motors 2 and 4 generate more thrust than motors 1 and 3.
- To increase altitude, all motors increase the thrust equally.

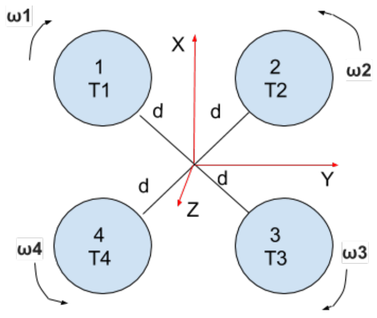


Fig. 1. Forces in a quadrotor

B. Gripper Design

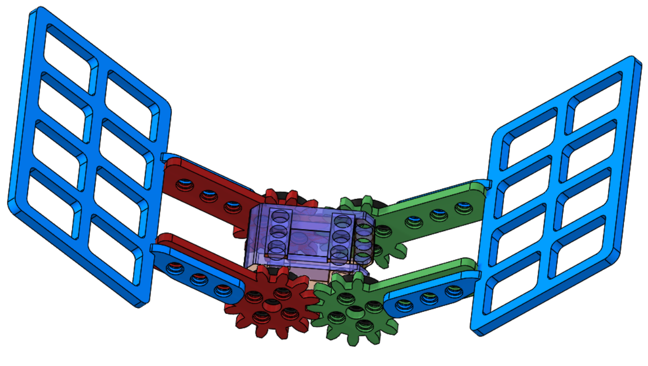


Fig. 2. Gripper Assembly

The gripper design was one of the crucial tasks for this project since a majority of the tasks involved picking/ dropping the blocks. Listed below are some of the considerations that we thought were important for the design selection: ω

i The gripper must be able to account for tolerances in all three axis as the quadrotor is constantly moving.

ii It has to be light weight, since the motor provides only limited torque.

iii It should be able to retract completely in idle position without obstructing the landing gear

After a few iterations, our final design is shown in Figure 2. We achieved consideration (i) by increasing the length of the gripping face to twice the side length of a block. We also had a large radius for the gripper movement, which helped in compensating for any errors in quadrotor position. Consideration (ii) was achieved by reducing the weight of the gripping arm by adding slots. When moved to negative angles of the servo, the gripping arm would completely be behind the motor and hence ensured that consideration (iii) was met.

C. Electronics Architecture

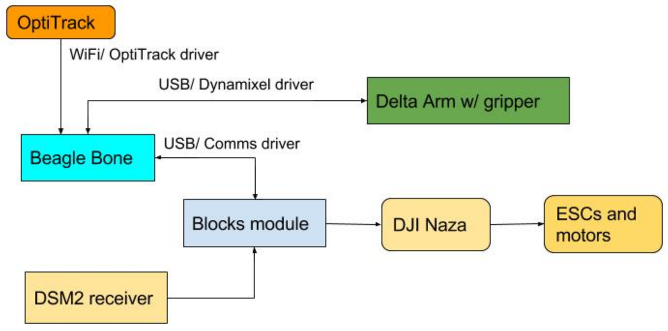


Fig. 3. Communication block diagram

All the algorithms required for controlling the quadrotor and manipulating the delta arm is run on a Beagle Bone Black (BBB), running Debain Linux. Figure 3 shows the interface between BBB and various sub systems that are used to observe or manipulate the state of the quadrotor-delta arm system. Functionality of each of the interfaces is summarized below.

i OptiTrack: OptiTrack, provides the real time pose $(x, y, z, \theta, \phi, \psi)$of the quadrotor, measured on world coordinate system. The setup consists of 6 IR cameras facing the region of interest. The objects to be tracked, (quadrotor, pick up point, drop off point) are affixed with multiple retro reflector markers. The cameras detect these markers and provide accurate pose of the object. More details of the OptiTrack system can be found at http://optitrack.com/products/prime-13/. The data from OptiTrack was handled by `optitrack_driver`, which published the three poses as LCM messages.

ii Blocks: Blocks module is used to select between autonomous control and manual mode. It also is used to read all the RC commands that are sent from the controller. When in autonomous mode, the blocks transmits the commands from BBB to the flight controller, DJI Naza. When in manual mode, it passes the RC signals seen by the DSM2 to DJI Naza. The incoming and outgoing data of block is handled by `comms_driver` over LCM messages.

iii Dynamixel Motors: The delta arm consisted of four Dynamixel XL-320[1] motors, three for controlling the (x, y, z) position and one motor for gripper. The servos were powered by 7.5V and were connected to BBB over USB to serial interface. The motor commands and statuses were handles by `dynamixel_driver` over LCM messages.

.

III. Delta Arm Kinematics

A. Delta Arm Design

TABLE I
Physical Constants of the Delta Arm

Name	Description	Value (mm)
s_B	base reference triangle side length	220
s_P	end effector reference triangle side length	66
L	Top link length	120
l	bottom link length	193

These values were measured by hand with a ruler, and are thus subject to error +/- 2mm and follow the the follow reference

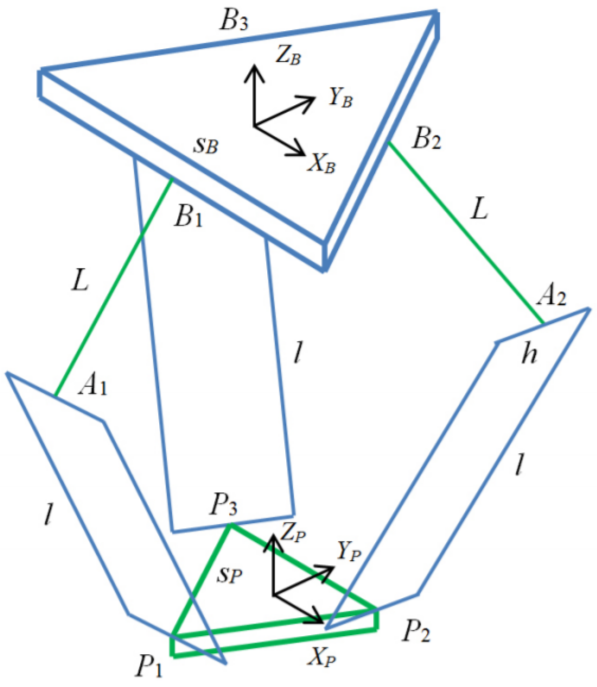


Fig. 4. Delta Arm Kinematic Description

B. Forward Kinematics

For forward Kinematics, the goal, given a set of actuated joint angles, is to determine the Cartesian position of the bottom moving platform origin. Although usually easier than

[1]The motors can be found at http://www.robotis.us/dynamixel-xl-320/

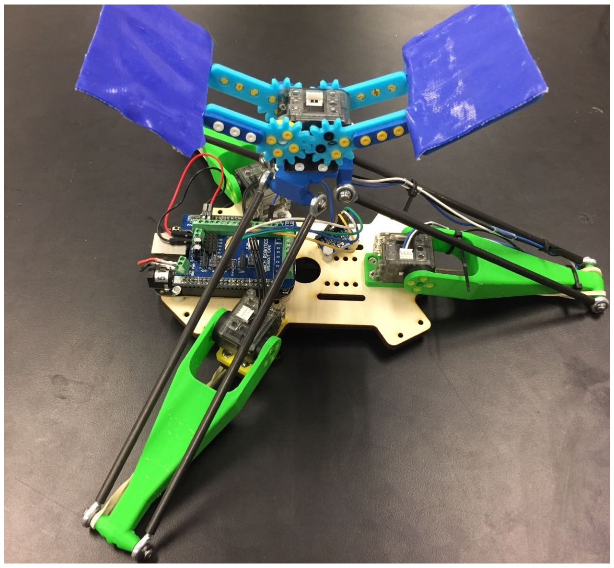


Fig. 5. Complete delta arm assembly

inverse kinematics, in the case of delta arm, there is considerable amount of foundation that needs to be laid in order to accomplish forward kinematics. For the purposes of our quad rotor, forward kinematics were used for validation of IK and testing purposes. They are not explicitly part of the end effector state machine.

Needless to say, to calculate the FK, variables must be defined for the the two frames of interest, the base and the end effector.

Taking the center of the base to be the origin, the following vectors are defined for the distance to the side edge of the platform.

$$B_1^B = \begin{bmatrix} 0 \\ -W_B \\ 0 \end{bmatrix} \quad B_2^B = \begin{bmatrix} \frac{\sqrt{3}}{2}W_B \\ \frac{1}{2}W_B \\ 0 \end{bmatrix} \quad B_3^B = \begin{bmatrix} -\frac{\sqrt{3}}{2}W_B \\ \frac{1}{2}W_B \\ 0 \end{bmatrix}$$

$$P_1^P = \begin{bmatrix} 0 \\ -W_B \\ 0 \end{bmatrix} \quad P_2^P = \begin{bmatrix} 0 \\ -W_B \\ 0 \end{bmatrix} \quad P_3^P = \begin{bmatrix} 0 \\ -W_B \\ 0 \end{bmatrix}$$

Then the distance to the vertices are defined

$$b_1^B = \begin{bmatrix} 0 \\ -W_B \\ 0 \end{bmatrix} \quad b_2^B = \begin{bmatrix} \frac{\sqrt{3}}{2}W_B \\ \frac{1}{2}W_B \\ 0 \end{bmatrix} \quad b_3^B = \begin{bmatrix} -\frac{\sqrt{3}}{2}W_B \\ \frac{1}{2}W_B \\ 0 \end{bmatrix}$$

where

$$w_B = \frac{\sqrt{3}}{6}s_B \quad u_B = \frac{\sqrt{3}}{3}s_B \quad w_P = \frac{\sqrt{3}}{6}s_P \quad u_B = \frac{\sqrt{3}}{3}s_P \tag{1}$$

Then the Kinematic equations are governed by these vector relations

$$\{B_i^B\} + \{L_i^B\} + \{l_i^B\} = \{P_P^B\} + \{P_i^P\} \tag{2}$$

The L vectors are given by

$$L_i^B = \begin{bmatrix} 0 \\ -L\cos\theta_1 \\ -L\sin\theta_1 \end{bmatrix}$$

$$L_2^B = \begin{bmatrix} \frac{\sqrt{3}}{2}L\cos\theta_2 \\ \frac{1}{2}L\cos\theta_2 \\ -L\sin\theta_2 \end{bmatrix}$$

$$L_3^B = \begin{bmatrix} -\frac{\sqrt{3}}{2}L\cos\theta_3 \\ \frac{1}{2}L\cos\theta_3 \\ -L\sin\theta_3 \end{bmatrix}$$

Similarly for the lower legs we have

$$l_i^B = \begin{bmatrix} x \\ y + L\cos\theta_1 + a \\ z + L\sin\theta_1 \end{bmatrix}$$

$$l_2^B = \begin{bmatrix} x + \frac{\sqrt{3}}{2}L\cos\theta_2 + b \\ y - \frac{1}{2}L\cos\theta_2 + c \\ z + L\sin\theta_2 \end{bmatrix}$$

$$l_3^B = \begin{bmatrix} x + \frac{\sqrt{3}}{2}L\cos\theta_3 - b \\ y - \frac{1}{2}L\cos\theta_3 + c \\ z + L\sin\theta_3 \end{bmatrix}$$

with these relations we can solve for the x,y,z coordinated, or the vector P_P^B

This can be done in a variety of ways, such as the method of virtual spheres at the knee joints.

C. Inverse Kinematics

The goal of inverse kinematics is to derive the values of θ given the Cartesian coordinators of the end effector. This dual case, is much more useful for our purposes. Since we are inputting a Cartesian coordinate for a block, and we are tracking the position of the quadrotor, the precise grasping location is a simple vector difference, then IK can be used to actually send an actuation signal. To Calculate IK, we first need to define some preliminary variables.

By defining these variables as such

$$E_1 = 2L(y + a)$$
$$E_2 = -L(\sqrt{3}(x + b) + y + c)$$
$$E_3 = L(\sqrt{3}(x - b) - y - c)$$
$$F_1 = 2zL$$
$$F_2 = 2zl$$
$$F_3 = 2zl$$
$$G_1 = x^2 + y^2 + z^2 + a^2 + L^2 + 2ya - l^2$$
$$G_2 = x^2 + y^2 + z^2 + b^2 + c^2 + L^2 + 2(xb + yc) - l^2$$
$$G_3 = x^2 + y^2 + z^2 + b^2 + c^2 + L^2 + 2(-xb + yc) - l^2$$

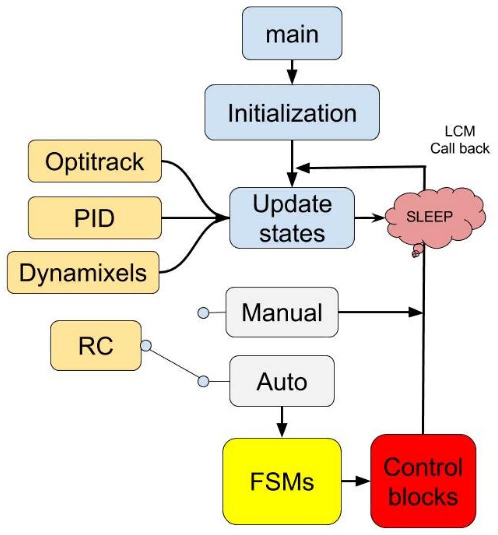


Fig. 6. Software Architecture.

Then three equations can be formed that the inverse kinematics must follow

$$E_i \cos\theta_i + F_i \sin\theta_i + G_i = 0 \quad \text{for i =1,2,3} \tag{3}$$

using the Tangent Half-Angle Substitution, we arrive at the needed equation for θ

$$\theta_i = 2 \arctan t_i \tag{4}$$

where t_i is defined by the the quadratic equation

$$t_{1,2} = \frac{-F_i \sqrt{E_i^2 + F_i^2 - G_i^2}}{G_i - E_i} \tag{5}$$

Even though this equation gives two solutions, we are only interested in the solution where the knees of the delta arm stick out. With this constraint, we are able to uniquely solve the IK for a given (x,y,x) coordinate.

IV. Software Design

A multi-thread software system is implemented for control of the Quadrotor and Deltaarm. The main thread is designed to achieve various kinds of tasks such as hovering and blocks transporting. Optional printing threads is integrated for debugging and to display in real-time robot state.

The main event-driven thread architecture is shown in Fig. 6. After device initialization, the robot falls into sleep to reduce power consumption and waits for interrupts. Four LCM channels are subscribed to update robot states or to execute motion control. Optitrack channel provides real-time robot position and target position (Pick up place & Drop off place). Another process is designed to detect keyboard input and send LCM messages to PID channel, from which BeagleBone interpret the messages to tune PID gains for motion control. Dynamixels channel is used to attain servo states feedback of deltaarm and gripper. RC channel receives messages from quadrotor control stick to decide running manual mode or auto mode. In manual mode, robot passes control pwm commands to NAZA block and falls into sleep. In auto mode, robot runs specific state machine and control blocks for motion controls of both quadrotor and robot arm. When switching from manual mode to auto mode, program record current position of quadrotor as setting hold point. When switch from auto mode to manual mode, robot reset delta arm pose hence it can avoid colliding with ground when landing off.

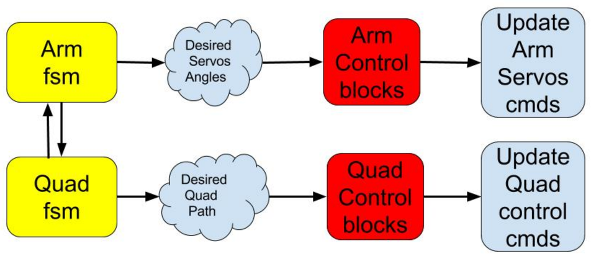


Fig. 7. Cooperative State Machine Framework.

A cooperative finite state machine framework, as shown in Fig. 7. This design not only reduces the difficulty in handling complex tasks such as block transporting, but also makes it much easier for debugging. Two cooperative state machines are implemented including the Arm fsm and Quadrotor fsm. Quadrotor state machines are responsible for quadrotor motion control depending on specific task. It generates desired path setpoints as input of quadrotor control blocks, which outputs quadrotor control pwm commands. Arm state machine is used to handle arm control such as blocks picking and placing. It generates desired deltaarm and gripper pose as input of arm control blocks, which outputs servos contorl pwm commands. Communication between two state machines are based on function call method. Quadrotor fsm keeps calling Arm fsm with a command parameter, Arm fsm executes based on the command value returns current state.

V. State Machine & Control System

In this section we will discuss an end-to-end robot control system from specific event to final control commands.

A. State Machines and Decision Making Architecture

In our design, the quadrotor robot is able to handle multiple events, including `HOVER`, `TRANSIT`, `HOVER`, `PICK&PLACE`, and `CHASE`. In the `HOVER` event, the only task is to hover at the setting hold point, which is recorded by system when switching from manually mode to auto mode. Hence state machine is not required for this event. For other events, correspondent state machines are designed.

1) Transit Event: In this event, the main task for the quadrotor is to hover in a place for 5 seconds, move to the setting destination point, stay for 5 seconds and then move back. An optimized state machine is designed as shown in

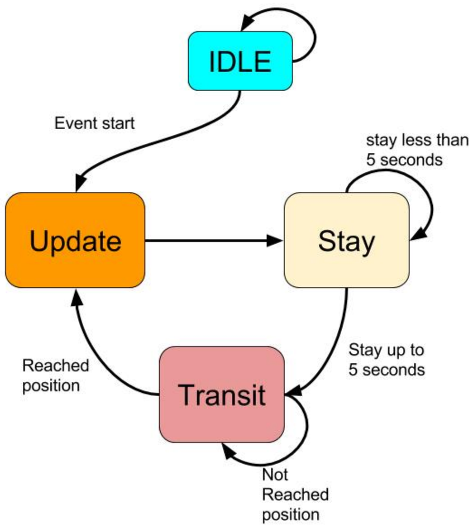


Fig. 8. Transit Event State Machine.

Fig. 8. Since no block operation is required in this event, arm state machine is unnecessary. Under this state machine, the quadrotor is able to keep transiting between hold point and destination point.

As shown in Fig. 8, there are four states included in Transit Event state machine:

`IDLE`: The robot waits for RC command to switch from manual mode to auto mode. This is the default state when the robot is switched from manual mode to auto mode. It records current quadrotor position as hold point and moves to `Update` state.

`Update`: This state is used to update desired path set points. A move direction token is used to identify if the robot is moving from hold point to destination point or on the contrary. Token value will be toggled in this state and be used in `Stay` and `Transit` state. `Stay`: In this state the robot hover in set point for 5 seconds. The set point is defined by the move direction token. If token is 0, the set point is set to the recored hold point; otherwise, it will be set to destination point. After 5 Seconds, state will be changed to `Transit` state. `Transit`: In this state robot moves from a begin point to an end point, which are defined by move direction token. If token is 0, begin point is set the record hold point and end point is set to destination point; otherwise, two points switch. Both distance and velocity are used to determine if the robot reach to end point. If both of them are less than setting threshold, end point is reached and state will be changed to `Transit` state.

2) Pick & Place Event and Chase Event: In these events, the main task for the quadrotor is to pick up blocs from the setting Pick Up place and drop blocks at Drop Off place. Pick Up place and Drop Off place are both updated from optitrack messages so two events are logically equivalent. Since the task is complex, cooperative state machines are used as discussed in the previous section.

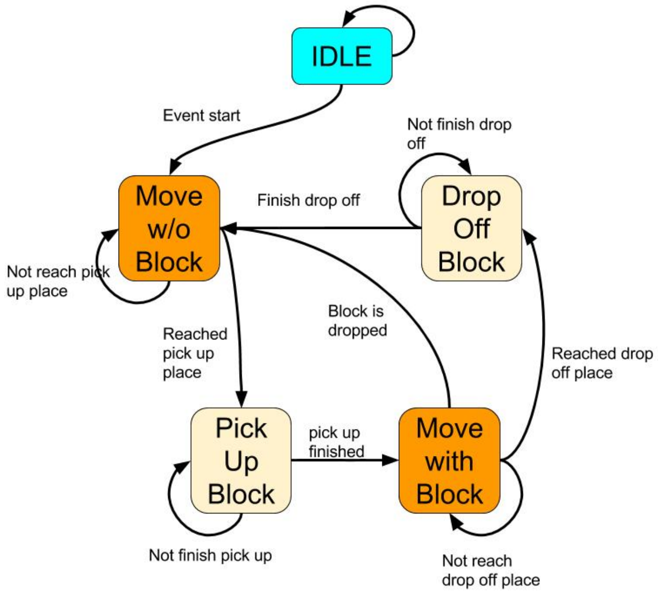


Fig. 9. Pick & Place Event Quadrotor State Machine.

quadrotor state machine: Quadrotor state machine is responsible for quadrotor movement decisions. As shown in Fig. 9, there are five states included in Pick & Place Event quadrotor state machine:

`IDLE`: The robot waits for RC command to switch from manual mode to auto mode. This is the default state when the robot is switched from manual mode to auto mode. It records current quadrotor position as hold point and moves to `Move W/o Block` state.

`Move W/o Block`: In this state the robot transit from hold point to Pick Up place without blocks in hand. Same as transit event, both distance and velocity error are used to determine if the robot reaches Pick Up place. If it reaches successfully, state is changed to `Pick Up Block` state.

`Pick up Block`: In this state, quadrotor fsm keep sending pick up command to arm fsm. And the quadrotor sets Pick Up place as hold point and hovers over there. Once the feedback from arm fsm denotes that the block is successfully picked up, the state will be changed to `Move With Block` state.

`Move With Block`: In this state robot transit from Pick Up place to Drop Off place while listening to arm fsm to determine if block is still in hand. If the feedback from fsm shows that block is dropped, the state would be changed to `Move W/o Block` so the quadrotor can go back to pick another block. If the quadrotor successfully reaches Drop Off place with the block, state will be moved to `Drop Off Block`

`Drop Off Block`: In this state quadrotor fsm keep sending drop off command to arm fsm. The quadrotor sets Drop Off place as hold point. In `CHASE` event, the stable velocity is

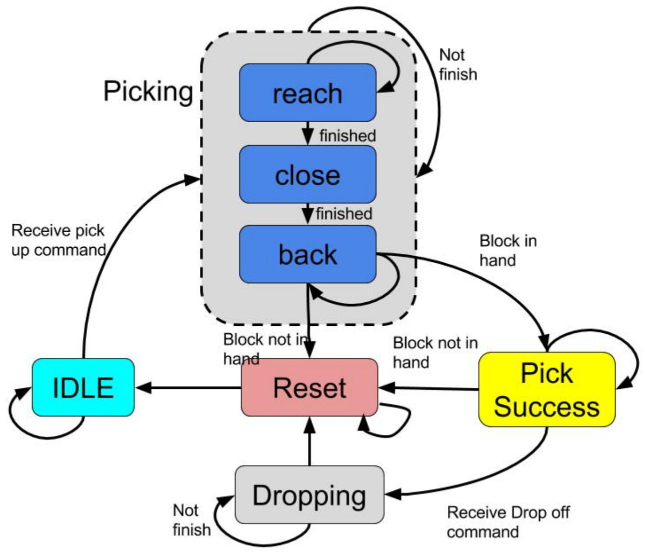


Fig. 10. Pick & Place Event Arm State Machine.

the velocity same as Drop Off place moving velocity. Once the feedback from arm fsm denotes that the block is successfully dropped off, the state will be changed to `Move W/o Block` state so the quadrotor can move back to pick another block.

Before the Arm fsm is implemented, feedback of Arm fsm can be set as ideal values so we can test and debug the transit functionality identically. With this state machine, the quadrotor can transit from Pick Up place and Drop Off place and keep track of Drop Off place as expected.

Arm state machine: Arm state machine is responsible for delta arm and gripper movement decisions. Current state of this state machine would be sent to Quadrotor state machine. As shown in Fig. 10, a two-layer state machine is used to denote states of the arm. There are five main states included in Pick & Place Event arm state machine and three sub-states in Picking state:

`IDLE`: Arm waits for quadrotor control command to start picking. This is the default state of the arm.

`PICKING`: This state is used to pick up a block. Three sub-states are used to represent the state of reaching the block, grasping the block, and moving the arm back. Three sub-states are taking place step by step. Once a fault occur such as the arm is no longer able to reach the block (no result from delta arm IK) or block is dropped while holding it back, the state switches to `RESET` state. If the delta arm moves back with block in hand, the state would be changed into `Pick Success` state.

`PICK SUCCESS`: In this state, the gripper keep grasping the block. This state denotes that the block is consistently in hand and the quadrotor can keep heading to Drop Off place. If feedback from gripper shows that no block is in hand, state will be changed to `RESET` state. Once the state machine receives message from quadrotor fsm to drop off the block, the state will be switched to `DROPPING` state.

`DROPPING`: This state is used to drop the block by opening the gripper.

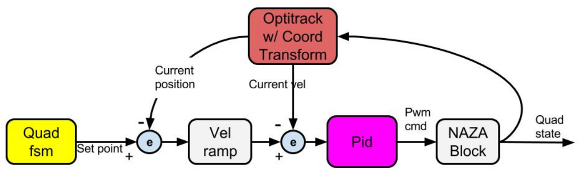


Fig. 11. flight control in one axis.

`RESET`: In this state, the delta arm is reset to 0,0,-0.1 in robot's coordination. And the gripper is set opened to prepare for next catch. After successfully reseting, robot changes the stateto `IDLE` state and waits for next pick up command from quadrotor fsm.

Before the Quadrotor fsm is implemented, command for Arm fsm can be set as ideal values so we can test and debug the pick up and drop off functionality identically. Combined with two state machines, the quadrotor can successfully finish pick & place event and chase event.

B. Control System

A quadrotor control system is designed for quadrotor motion control as shown in Fig. 11 The control block structure is identical in three axis. Input of the block is the set points decided by quadrotor state machine. Based on the current position and given set points, desired moving velocity can be calculated by velocity ramp function, which is shown in Fig. 12. Parameter of ramp function is shown in Tab. III. Current position and velocity of quadrotor is given by Optitrack navigation system. Since coordinate between quadrotor and optitrack is different, coordinate transform is required before using the data. The error of desired velocity and current quadrotor velocity is used as input of PID controller. The output of the PID controller is the weighted sum of these three terms:

$$u(t) = Base + K_p e(t) + K_i \int_0^t e(t)dt + K_d \frac{de(t)}{dt} \tag{6}$$

PID gains are tuned appropriately by PID process as shown in Tab. 2. Output of PID controller is the pwm commands for NAZA block. Saturation is set to a reasonable value in case the quadrotor moves brutally.

For gripper and delta arm control, a torque feedback based block detection algorithm is designed to determine the gripper status as shown below by using both torque and angle feedbacks.

TABLE II
CONTROL GAINS TABLE

Controllers	P	I	D	$BASE$	MAX	MIN
Thrust PID	1000	0	0	1500	1700	1480
Pitch PID	1000	0	0	1500	1650	1450
Roll PID	1000	0	0	1500	1650	1450

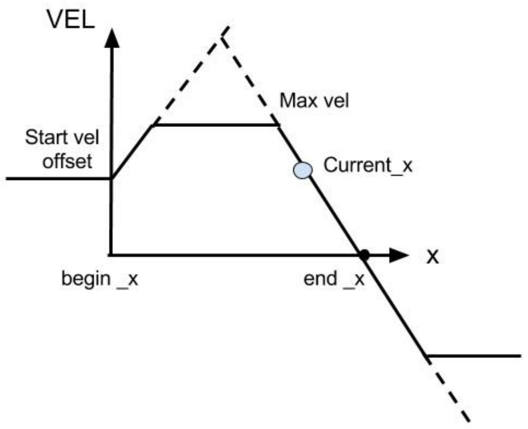


Fig. 12. velocity ramp function.

TABLE III
VELOCITY RAMP FUNCTION PARAMETER

Name	Description	Value (m/s)
s_o	start velocity offset	0.2
s_l	ramp function slope	0.8
Max	velocity saturation	0.5

Algorithm 1 Torque feedback based block detection

```
Input: Servo 4: command position radians, status position radians, status load
Output: Gripper status
1: first statement
2: if (command position radians-status position radians $<$ tolerance) then
3:     gripper status = no block
4: else if status load$\geq$Threshold then
5:     gripper status = block grabbed
6: else
7:     gripper status= in motion
8: end if
```

VI. Data Acquisition and Performance Analysis

A. PID Tuning and Performance

Using the software implementation of changing controller gain values on the fly, we were able do tuning mid flight. This helped tremendously, given the limited access to flight time. It seemed that carefully choosing saturation limits greatly affected the performance of our controller. After reviewing the block output data, it was apparent that this was the case due to our high gain values, making our controller essentially a bang-bang implementation. However, it did provide robust stability, and efforts to lower gains to normal pid loop behavior resulted in decreased performance. Since time was limited, and we did not see adverse effects with our implementation, we continued and successfully used our controller in the competition with large gains and fine tuned saturation limits.

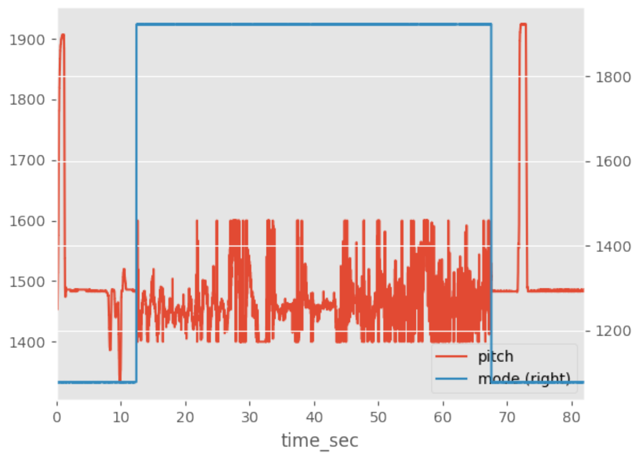


Fig. 13. Pitch Controller Output for Hover in Place

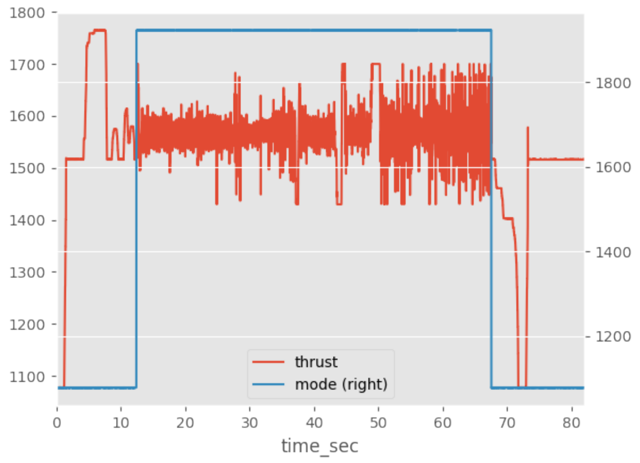


Fig. 14. Thrust Controller Output for Hover in Place

B. Results

1) Hovering: The task of hovering in place , went smoothly with no issues. This was a direct test of our controller, which we tested rigorously. We were able to reach a steady state x variance of .022m , y variance of .130m , and z variance of .0002m. It is interesting to note that the y coordinate experienced more disturbance than the x. Since the the roll and pitch controllers are identical and the quadcopter itself is symmetrical, this would not normally be expected.

2) Transit: This task was also went smoothly. With a movement and settle time to within 20% of our hover steady state of 2.016 seconds, this again showed the robust and responsive controller used. The z coordinate can be seen moving between waypoints in this figure.

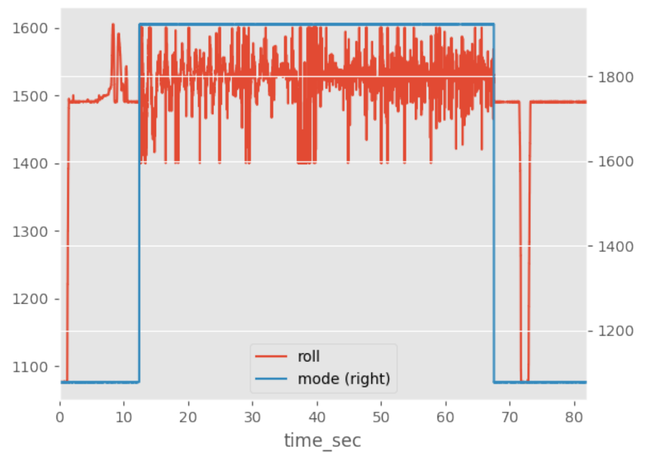


Fig. 15. roll Controller Output for Hover in Place

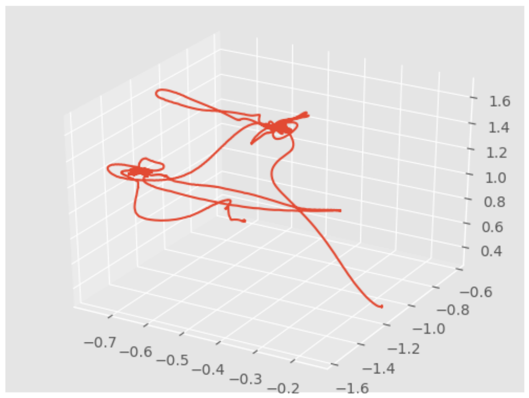


Fig. 17. Transit Task Cartesian Trajectory

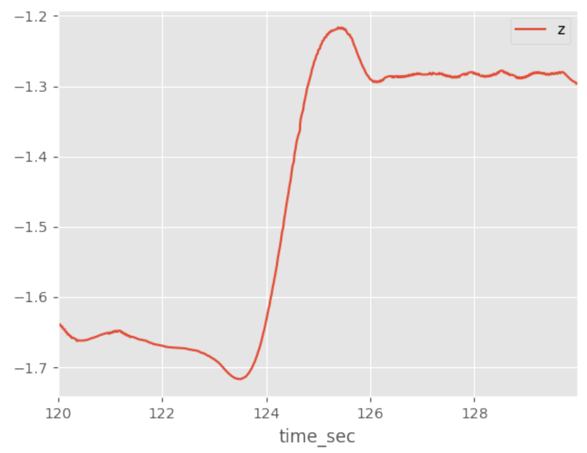


Fig. 16. Transit z Response

However on the first trial, the quadcopter drove straight upwards with no control. As can be seen from the plot, this was almost certainly an optitrack error in which caused this malfunction. With additional time, it would have been wise to add logic in the state machine for loss of optitrack data, so that sporadic and dangerous velocity commands are not given.

3) Pick & Place: The Pick and Place task was completed on the first attempt, in which we successfully flew to a pick up location grasped a block with the delta arm, then flew to the drop off location and dropped the block into a bucket.

4) Pick & Place: Several issues related to delta arm actuation and grasping caused numerous failed attempts. In the initial trial, An offset could be seen between the quadrotor and the drop off location, interestingly, this did not deter success in task 3. This offset may be because the rigid body create by optitrack does not have the the same center point as the actual quadrotor. Nevertheless, with a simple offset added to the location of the pick up waypoint , we were able to achieve precise position over the target for delta arm actuation. Nearly all other issues related to task 4 performance were related to the delta arm and state logic associated, which is discussed in detail in the improvement section. Although we did not complete all blocks, we did complete more than half of the attempts.

5) Chase: Task 5 was partially completed. The block was able to be picked up, as would be expected from the performance of task 3 and 4, but was not able to drop off the block into the moving the target. We were able to successfully track and follow the drop off location, but we were not a ble to drop the block into the bucket. Due to limited trials and a drained battery, it is not completely apparent all corrections needed, but it was obvious that the quadrotor did not move fast enough to reach the moving target. Some issues with the bucket itself also added difficulties, since it was not secured, the bucket would tip to its side, which caused an unsuccessful drop, since the bucket was no longer directly under the drop off rigid body center.

VII. Improvements

Related to delta arm actuation there are several areas for improvement. As seen in the results, there was about 50% success rate in the performance of pick up and place. One issue is related to the logic that determines if a block was successfully grabbed. We used a combination of angle, torque and time stationary to determine if the gripper was actually on a block, which would then change the state to transit. Many attempts led to the gripper latching onto the base of the pickup pole. This would then pass the logic as successful, but when the quad attempted to lift up, the immediate drop in torque from slipping off the base to the block would trigger the zero torque feedback and immediately open up the gripper, thus completely dropping the block and failing the attempt. To alleviate this issue, it would help to fine tune the angle in the logic. Since the base is wider than the block, it would create a grip angle that would necessarily be larger than any angle produced from actually grabbing the block. In the ideal case then, the gripper would sense a bad grasp, open and retract, then proceed to attempt again without touching the block in this process.

Another area for improvement would be the velocity limit of our controller to follow a moving target, as in the case of task 5. For stability of our controller, we impose max limits of velocity in the x,y,z coordinates. However, this was not able to follow the speed of the RC controlled bucket. Furthermore it was not able to handle the case in which the bucket reversed direction immediate, in which case our controller would sense it was directly under the bucket and then proceed to drop the block, but would then miss the target. Since we didn't know the speed of the RC bucket ahead of time, it would be difficult to optimize a speed that would allow following and maintain our levels of stability, but the other issue could be fixed by including logic to that would require the target to be within a delta region for a certain amount of time. For example, if it was required to be directly under the bucket for two seconds before release, this would eliminate the case where the quad would drop the block as soon as the RC bucket reversed direction.

Related, since we are try to track a moving target , is would be ideal to match velocity with the target and achieve zero steady state error in this case. However, since our controller does not have an integrator, this goal is inherently unachievable without an offset. Since the RC bucket velocity is unknown, an offset is not practical, thus It would be wise to add an integrator to the controller we used. Since our controller is based on velocity and not position, a single integrator would suffice to achieve zero steady state error in velocity.

Another issue related to grasping is that even with visually good grasps on the block, it was still the case that a large portion of attempts would end with the gripper releasing the block in the process of retracting to the inside position. This seems to hint at a mis-calibration of our torque logic. It should be noted that the servos used in conjunction to the gripper design gave many instances of torque lockup. In testing, we tried to optimize the grip strength and be able to withstand at least 30 seconds before the motors would lock up. After seeing the speed at which we were able to drop off blocks, 30 seconds seems to be too conservative, and it would have helped to use more torque, which would then help create clearer logic to determine if the block was actually grabbed or not.

VIII. Conclusion

We Successfully implemented and tuned an outer loop controller to maintain the desired position of a quadrotor in cartersian world coordinate system (X, Y, Z). We were able to sucessfully transit between way points, while picking up blocks at one way point and dropping at another. We could chase the bucketbot, while trying to drop the block, but could not accurately track the target within a small enough neighborhood to drop the block in the bucket. Given more time, we would be able to improve the performance of our control algorithm and complete all the tasks. During the competition day we attempted all the tasks and scored 52 points. Our team was placed 4th in the competition.

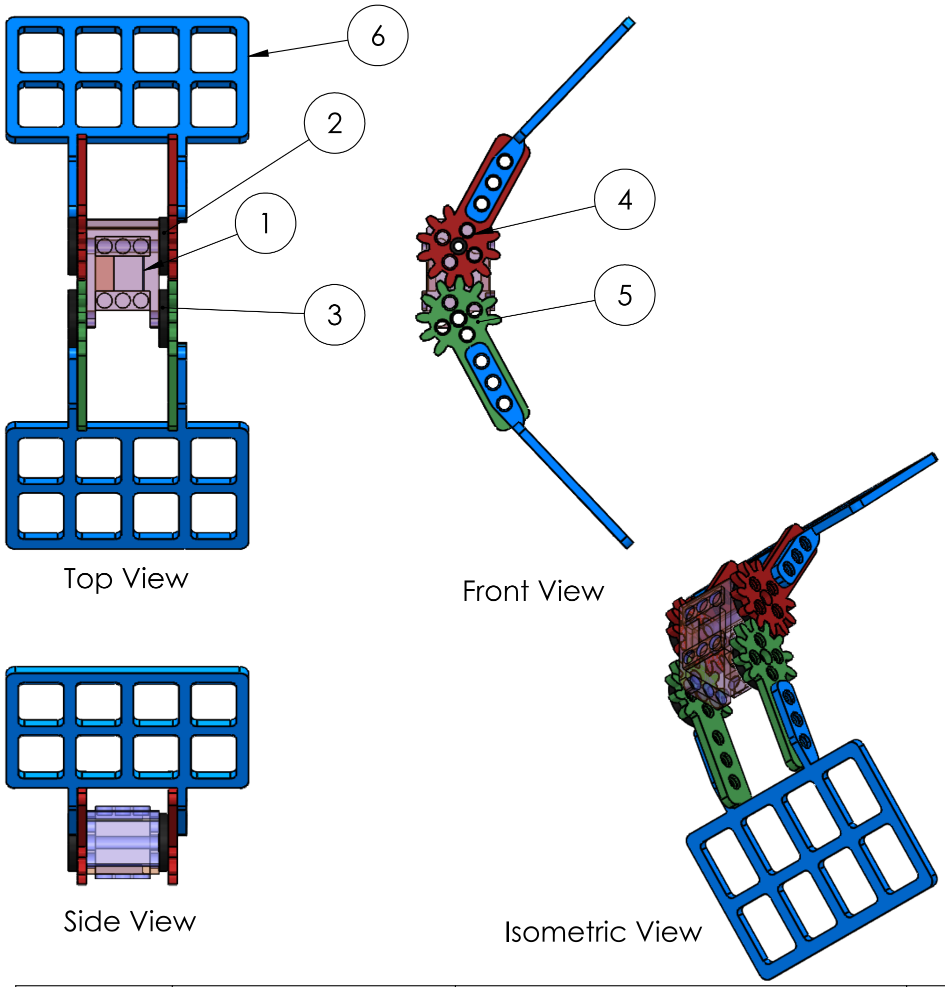
Acknowledgment

We would like to thank Prof. Ella Atkins, Dr. Peter Gaskell, Mr. Theodore Nowak, Mr. Abhiram Krishna, our fellow students in ROB 550 course for the helping and inspiring us through out the duration of the project. We are grateful to the Robotics Institute, University of Michigan for providing us with all the resources needed for the project.

References

[1] Spong, Mark W., Seth Hutchinson, and Mathukumalli Vidyasagar. Robot modeling and control. Vol. 3. New York: wiley, 2006.
[2] Sebastian Thrun, Wolfram Burgard, Dieter Fox. Probabilistic Robotics (Intelligent Robotics and Autonomous Agents series)
[3] http://www.ohio.edu/people/williar4/html/pdf/DeltaKin.pdf

Appendix A
Bill of Material: Gripper



ITEM NO.	PART NUMBER	DESCRIPTION	QTY.
1	XL320_Servo	Dynamixel XL-320 Servo	1
2	XL320_Hub	OEM part	1
3	XL320_Idler	OEM part	3
4	Lgear_hole_offset		2
5	Rgear_hole		2
6	grip		2
7	Ollo Rivets	OEM Part	28

DWG NO.
Gripper_assembly A4
SHEET 1 OF 1