Decentralized Cooperation with Actor-Critic Algorithms Driving Multi-Agent Mission Planning

Victoria Nagorski

***Abstract*—“Decentralized Cooperation with Actor-Critic Algorithms Driving Multi-Agent Mission Planning” goes into depth on how to setup a Simulink model for a multi-agent task. The objective of the task is for three 1 kg quadcopters to carry a 0.9 kg object through an obstacle. This paper covers all of the assumptions made, how to setup the Simulink model, what difficulties were faced, and where improvements can be made for a higher fidelity model.**

I. Introduction

A. Problem and Motivation

Small quadcopters, commonly called “drones” in the public market, are becoming increasingly popular. One might see them used in aviation research, military/law enforcement, search and rescue, order deliveries, transporting people, mining, news and sports, surveying and construction, and reality games [1]. Why is there this increasing popularity for quadcopters? As far as being easily accessible, quadcopters are affordable, mechanically simple, easy to operate and control, safe with small blades, and can explore hard to reach areas [2,3,4].

For the reasons listed above, quadcopters are also a prime surrogate to use with multi-agent networks. Quadcopters by themselves cannot efficiently perform tasks such as carrying large loads due to their innability to scale up well [3]. A team of quadcopters could be used to achieve a task, such as carrying a heavy load, that would not normally be possible for the individual agent. However, one has to be careful to use a decentralized approach in their methods. Centralized methods can suffer from being more computationally heavy when compared to their decentralized counterparts [5]. It is also worth noting that decentralized methods are more robust to agents being added or dropped to their network [5].

With advances being made in the cooperative engagement of quadcopters, more applications are being developed. One quadcopter might not be able to lift a 1 lb object, but three might. This capability could be used to deliver packages at people’s door-step, or deliver medicine and supplies to those in hard to reach areas.

For this project, the model is based off of a F330 body frame series. The arm length of this quadcopter is 155 mm, and the total weight of this quadcopter is approximately 1 kg [13]. For this model, the mass will be set to exactly 1 kg. These specifications are based off the quadcopter design used in paper [5]. Below is a general photo of what this specific quadcopter model looks like:

Fig. 1. Sample of an F330 body frame.

A group of these quadcopters will need to work together to carry 0.9 kg object through an obstacle. These quadcopters will be receiving regular intervals of communication from each other. However, they will need to overcome sources of error from noise in the sensors themselves. To do this, a reinforcement learning (RL) algorithm called MADDPG will be used to replace the optimal control algorithm used in [5] to navigate the quadcopters through an obstacle. This differs from previous MADDPG work because this problem focuses on a 3-D state instead of a 2-D space, has to take into considerations aircraft stability and limitations, and work through sensor noise. The MADDGP algorithm in this paper is not acting directly as the controller, but instead as the mission planner to feed the reference signal into the controller. This is distinction that this project makes that other papers do not emphasize. This project will also attempt to integrate the model using only Matlab's and Simulink's tools. Matlab is considered an industry standard in the GNC world, so it made sense to test how far the project could go in the platform.

B. *Outline of Paper*

The rest of this paper will follow the following structure: quadcopter design, multi-quadcopter problem formulation, simulation setup, main results, and conclusions. Quadcopter design's purpose is to take the reader through how the quadcopter model was built, and all the assumptions that went into its design. The research in this paper is all based off a model's performance, and thus it is important to capture all the design decisions made. Simplifications in a model could potentially produce different results when compared to a higher fidelity model.

The multi-quadcopter problem formulation section is supposed to capture the algorithms used, and the structure of the Vehicle Management System (VMS). The VMS includes everything from the mission planner to the controller. Before going into the main results, the simulation set-up will be presented, and then the main results will cover notable features of the Simulink model. Conclusions will discuss contributions and where further work can be done to improve this model.

II. Quadcopter Design

This section captures my attempt to produce a reasonably realistic model of a quadcopter within the time constraints of a semester. This means that some of the values used may be estimates, and can be improved with the use of an actual quadcopter. For example, I assumed that the quadcopter was symmetric about the x, y, and z axis. This allowed me to set the products in the inertial matrix to 0. With more time, I could have modeled the quadcopter with all of its payloads in CAD to get a better estimate of the inertial matrix.

As the engineer of this quadcopter, I will have the following design freedoms:

- Initial positions of the quadcopters
- Trim position (what point to linearize about)
- Operational space of the quadcopters (altitude, how windy, etc.)
- Efficiency of the propellers
- Mass of the quadcopters
- Sensors (how many, how noisy, etc.)
- Level of symmetry in the quadcopters

The following assumptions were made in the model to make the problem easier to solve:

- The quadcopters are symmetric along the x, y, and z axis
- Rigid body
- Air flow is incompressible
- For estimating the inertia matrix, the mass is equally distributed throughout the body
- All states are measurable or solvable
- The sensors will be afflicted with different types of error that can be simply modeled by white noise
- Wind in this model is negligible to simplify the problem.
- The efficiency of the quadcopter is assumed to be 1
- Assuming that each motor can produce 10 N through flight- even with depreciating battery

A. *Quadcopter Kinematics [7, 10, 12]*

First, we will derive the forces that normally act upon the quadcopter. F_1, F_2, F_3, and F_4 are inputs into the system that we can control.

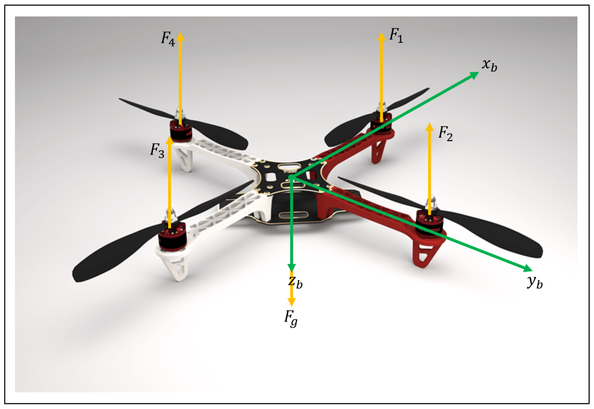

Fig. 2. Forces acting on quadcopter.

The first thing to notice is how the axis are oriented- it is typical to have the z-axis point "downwards". We will also notice that forces F_1, F_2, F_3, and F_4 do not act through the center of mass of the object. Therefore these forces will be creating moments about the center of mass. Before we derive the moments, we will define the force vector acting on the quadcopter with respect to the inertial frame. To start, we define the following rotation matrices:

$$C_3 = \begin{bmatrix} cos(\psi) & sin(\psi) & 0 \\ -sin(\psi) & cos(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$C_2 = \begin{bmatrix} cos(\theta) & 0 & -sin(\theta) \\ 0 & 1 & 0 \\ sin(\theta) & 0 & cos(\theta) \end{bmatrix}$$

$$C_1 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos(\phi) & sin(\phi) \\ 0 & -sin(\phi) & cos(\phi) \end{bmatrix}$$

Where:

$$\begin{bmatrix} \phi \\ \theta \\ \psi \end{bmatrix} = \begin{bmatrix} bank\ angle \\ pitch\ angle \\ yaw\ angle \end{bmatrix}$$

Then we can combine them for a 3-2-1 coordinate change:

$$C_N^B = C_3 C_2 C_1 \tag{1}$$

C_N^B takes the body frame into the inertial frame. Force gravity acts along the z-axis in the inertial frame. F_1, F_2, F_3, and F_4 all act along the $-z_b$ axis of the body frame attached to the quadcopter. We can write the first three equations in the inertial frame as:

$$m \begin{bmatrix} \ddot{x} \\ \ddot{y} \\ \ddot{z} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ mg \end{bmatrix} + C_N^B \begin{bmatrix} 0 \\ 0 \\ -(F_1 + F_2 + F_3 + F_4) \end{bmatrix} \tag{2}$$

We can then derive the next three equations of motion in the body frame:

$$\vec{M} = I\dot{\vec{\omega}} + \vec{\omega} \times (I\vec{\omega}) \tag{3}$$

Where the body rate vector can be defined as:

$$\vec{\omega} = \begin{bmatrix} p \\ q \\ r \end{bmatrix} = \begin{bmatrix} roll\ rate \\ pitch\ rate \\ yaw\ rate \end{bmatrix}$$

The inertial matrix of the quadcopter needs to be derived at this point. I will be defined as:

$$I = \begin{bmatrix} I_{xx} & I_{xy} & I_{xz} \\ I_{yx} & I_{yy} & I_{yz} \\ I_{zx} & I_{zy} & I_{zz} \end{bmatrix}$$

Because the quadcopter was assumed to be symmetric, the following is true:

$$I_{xy} = I_{yx} = I_{zx} = I_{xz} = I_{yz} = I_{zy} = 0$$

This means we only need to define I_{xx}, I_{yy}, and I_{zz}:

$$I_{xx} = \sum (y_i^2 + z_i^2) m_i$$

$$I_{yy} = \sum (x_i^2 + z_i^2) m_i$$

$$I_{zz} = \sum (x_i^2 + y_i^2) m_i$$

These values were taken from paper [12], and are summarized in table 1 in the linearization section. The placement of the payloads on the quadcopter is why $I_{xx} \neq I_{yy}$.
Figure 3 shows the quadcopter from a top view, and the important relationships needed to derive the moments acting on the quadcopter. $\vec{M}$ will be defined as:

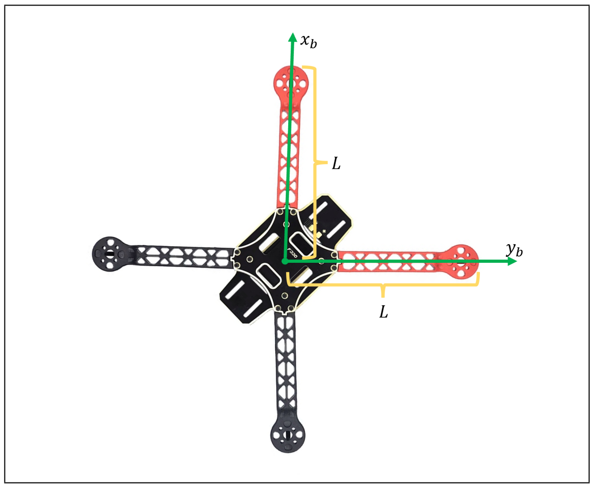

Fig. 3. Sample figure caption.

$$\vec{M} = \begin{bmatrix} L(F_4 - F_2) \\ L(F_1 - F_3) \\ M_1 - M_2 + M_3 - M_4 \end{bmatrix}$$

The last three equations that we will use relate the body rates to the Euler angles' rate of change.

$$\begin{bmatrix} p \\ q \\ r \end{bmatrix} = C_1^{-1} C_2^{-1} \begin{bmatrix} 0 \\ 0 \\ \dot{\psi} \end{bmatrix} + C_1^{-1} \begin{bmatrix} 0 \\ \dot{\theta} \\ 0 \end{bmatrix} + \begin{bmatrix} \dot{\phi} \\ 0 \\ 0 \end{bmatrix}$$

Which can be rewritten as:

$$\begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} 1 & sin(\phi)tan(\theta) & cos(\phi)tan(\theta) \\ 0 & cos(\phi) & -sin(\phi) \\ 0 & sin(\phi)sec(\theta) & cos(\phi)sec(\theta) \end{bmatrix} \begin{bmatrix} p \\ q \\ r \end{bmatrix} \tag{4}$$

Equations 2, 3, and 4 will be used to describe the equations of motion of the quadcopter.

B. Aerodynamics [8, 9]

There are two important equations that will play a role with the relationship of the propeller and the acting forces and moments. They are:

$$F_i = K_T \omega_i^2 \tag{5}$$

$$M_i = K_D \omega_i^2 \tag{6}$$

Where K_T and K_D are the thrust and drag torque coefficient respectively. ω_i represents how fast the propeller is spinning in radians on each arm. We can define the following variable:

$$\gamma = \frac{K_D}{K_T} = \frac{M_i}{F_i}$$

This relationship allows us to rewrite the $\vec{M}$ in equation 3 in terms of the inputs as appose to moments.

$$\vec{M} = \begin{bmatrix} L(F_4 - F_2) \\ L(F_1 - F_3) \\ \gamma(F_1 - F_2 + F_3 - F_4) \end{bmatrix}$$

C. Linearizing the Nonlinear Model

Before going into the linearization of the model, table 1 will be used to describe the numerical quantities for variables defined in the previous section. Next, we will define the states that we will be linearizing about. First we make $\dot{x} = x_1$, $\dot{y} = x_2$, and $\dot{z} = x_3$:

$$x = \begin{bmatrix} x_1 & x_2 & x_3 & p & q & r & \phi & \theta & \psi \end{bmatrix}^T$$

TABLE I
NUMERICAL QUANTITIES

Quadcopter		
Description	Variable	Value
Mass	m	1 kg
Gravity Acceleration	g	9.8 m/s^2
Propeller Efficiency Ratio	γ	1
Inertia Matrix [kg $\cdot m^2$]	I	$\begin{bmatrix} 0.01151 & 0 & 0 \\ 0 & 0.01169 & 0 \\ 0 & 0 & 0.01275 \end{bmatrix}$
Arm Length	L	0.155 m

Linearization through Taylor's expansion utilizes perturbations around an equilibrium point to approximate a linear model. The variables can be described as $x = x_e + \delta x$ and $u = u_e + \delta u$. These equations say that the state and inputs are an accumulation of both the equilibrium point and the small perturbations around that equilibrium. Taylor's expansion is the following:

$$\frac{d}{dt}x = f(x_e + \delta x, u_e + \delta u) = f(x_e, u_e) + \frac{\partial f}{\partial x}(x_e, u_e)\delta x + \frac{\partial f}{\partial u}(x_e, u_e)\delta u + H.O.T.$$

The differentials can then be lumped into the following matrix form (Jacobian matrices):

$$A = \frac{\partial f}{\partial x}(x_e, u_e) = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial x_1} & \cdots & \frac{\partial f_n}{\partial x_n} \end{bmatrix}$$

$$B = \frac{\partial f}{\partial u}(x_e, u_e) = \begin{bmatrix} \frac{\partial f_1}{\partial u_1} & \cdots & \frac{\partial f_1}{\partial u_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial u_1} & \cdots & \frac{\partial f_n}{\partial u_n} \end{bmatrix}$$

The end result is $\frac{d}{dt}\delta x = A\delta x + B\delta u$. The equilibrium state vector is defined by the engineer:

$$EquilibriumStateVector = x_e = \begin{bmatrix} x_{1e} \\ x_{2e} \\ x_{3e} \\ p_e \\ q_e \\ r_e \\ \phi_e \\ \theta_e \\ \psi_e \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

To solve for u_e, we need to set $\dot{x}$ equal to 0, plug in x_e, and then solve for u_e. Matlab solved u_e to be:

$$u_e = \begin{bmatrix} u_{1e} \\ u_{2e} \\ u_{3e} \\ u_{4e} \end{bmatrix} = \begin{bmatrix} 2.450 \\ 2.450 \\ 2.450 \\ 2.450 \end{bmatrix}$$

From there, we can derive the A and B matrices.

$$A = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & -9.8 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 9.8 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1.0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1.0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1.0 & 0 & 0 & 0 \end{bmatrix}$$

$$B = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ -1.0 & -1.0 & -1.0 & -1.0 \\ 0 & -13.467 & 0 & 13.467 \\ 13.259 & 0 & -13.259 & 0 \\ 78.431 & -78.431 & 78.431 & -78.431 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

We can decouple this matrix if we re-arrange the states as so:

$$x = \begin{bmatrix} x_1 & x_2 & x_3 & p & q & r & \phi & \theta & \psi \end{bmatrix}^T \rightarrow \begin{bmatrix} x_1 & q & \theta & x_2 & p & \phi & r & \psi & x_3 \end{bmatrix}^T$$

We get the following equations:

$$\begin{bmatrix} \ddot{x} \\ \dot{q} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} 0 & 0 & -9.8 \\ 0 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} \dot{x} \\ q \\ \theta \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 & 0 \\ 13.26 & 0 & -13.26 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} u \quad (7)$$

$$\begin{bmatrix}\ddot{y}\\ \dot{p}\\ \dot{\phi}\end{bmatrix} = \begin{bmatrix}0 & 0 & 9.8\\ 0 & 0 & 0\\ 0 & 1 & 0\end{bmatrix}\begin{bmatrix}\dot{y}\\ p\\ \phi\end{bmatrix} + \begin{bmatrix}0 & 0 & 0 & 0\\ 0 & -13.47 & 0 & 13.47\\ 0 & 0 & 0 & 0\end{bmatrix} u \quad (8)$$

$$\begin{bmatrix}\dot{r}\\ \dot{\psi}\end{bmatrix} = \begin{bmatrix}0 & 0\\ 1 & 0\end{bmatrix}\begin{bmatrix}r\\ \psi\end{bmatrix} + \begin{bmatrix}78.43 & -78.43 & 78.43 & -78.43\\ 0 & 0 & 0 & 0\end{bmatrix} u \quad (9)$$

$$\begin{bmatrix}\ddot{z}\end{bmatrix} = \begin{bmatrix}0\end{bmatrix}\begin{bmatrix}\dot{z}\end{bmatrix} + \begin{bmatrix}-1 & -1 & -1 & -1\end{bmatrix} u \quad (10)$$

The controller section will go through how to use dynamic inversion (DI) to control each individual subsystem within each quadcopter.

D. Sensors

For this paper, it will be assumed that all states can be measured. Therefore, the C matrix will be an identity matrix of size 6x6. The sensors to measure these states will be:

- Accelerometer
- Gyroscope
- GPS

Accelerometers are used determine the lateral acceleration of the quadcopter [11]. The accelerometer will give the states $\dot{u}, \dot{v},$ and $\dot{w}$. Gyroscopes, however, measure the rotational acceleration of the quadcopter [11]. The values $\dot{p}, \dot{q},$ and $\dot{r}$ come from the gyroscope. The GPS sensor will give the location of the quadcopter with respect to the earth based frame [12]. Having this sensor is important for performing tasks such as navigation. The sensors will be sampling at a rate of $100\ Hz$ or $T = 0.01\ s$.

To get $u, v, w, p, q,$ and r, the values from the sensors need to be integrated from their initial value. The Euler angles can be solved for by using the relationships in equation 4, and then integrating.

In real world scenarios, the sensors will provide the following sources of error:

- White Guassian noise
- Random (but fixed) bias and gain error
- Alignment error
- Latency and bandwidth limitations

For simplicity, the sensors will only add white noise in the readings. Simulink has a block called Band-Limited White Noise where it produces a continuous signal that varies with the sample rate specified. This block's sampling rate will match the sensor's sampling rate. Each value in the sensor's vector will have a separate noise block attached to it. The noise power will vary between $0.06 - 0.09$ for each sensor. The noise vector will be written as:

$$n = \begin{bmatrix}n_1 & n_2 & n_3 & n_4 & n_5 & n_6\end{bmatrix}^T$$

With a D_2 matrix:

$$D_2 = eye(6)$$

E. Controller Design

For the design of the quadcopter's controller, dynamic inversion (DI) was used to do inner and outer loop control. The inner loop is used to essentially stabilize the system while the outer loop controls the trajectory of the quadcopter.

The control loop structure this controller provides is shown in the figure below:

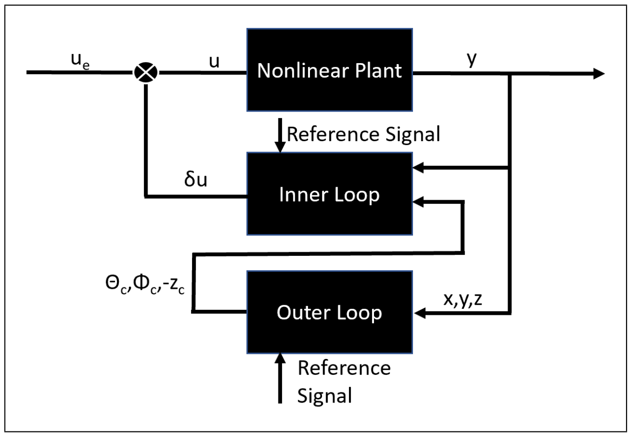

Fig. 4. Inner and outer loop control of quadcopter.

To derive the controllers, we will start with the concept of $\dot{x} = Ax + Bu$, and re-arrange it to get $u = B^{-1}(\dot{x}_{desired} - Ax)$. If we look at the A matrices and B^{-1} matrices of this system, we will see that $B^{-1}Ax = 0$ so we can write $u = B^{-1}(\dot{x}_{desired})$. This u equation will be important to solve for the u to input into the system.

The next step is to write out a second-order transfer function:

$$Y = \frac{\omega^2}{s^2 + 2\zeta\omega s + \omega^2} Y_c$$

Focusing on the inner loop, Y_c will be the variable that we can control given the B matrix. Using equation 7 and using θ as Y_c we will get:

$$\dot{q} = \omega^2(\theta_c - \theta) - 2\zeta\omega q \quad (11)$$

θ and q are measurable while θ_c is commanded. Using equation 8 and using ϕ as Y_c we get:

$$\dot{p} = \omega^2(\phi_c - \phi) - 2\zeta\omega p \tag{12}$$

ϕ and p are measurable while ϕ_c is commanded. Using equation 9 and using ψ as Y_c we get:

$$\dot{r} = \omega^2(\psi_c - \psi) - 2\zeta\omega r \tag{13}$$

ψ and r are measurable while ψ_c is commanded. Using equation 10 and using z as Y_c we get:

$$\ddot{z} = \omega^2(z_c - z) - 2\zeta\omega\dot{z} \tag{14}$$

z and $\dot{z}$ are measurable while z_c is commanded. Moving onto the outer loop to control the trajectory, we can use x as Y_c and write:

$$\ddot{x} = \omega^2(x_c - x) - 2\zeta\omega\dot{x} \tag{15}$$

x and $\dot{x}$ are measurable while x_c is commanded. Next we use y as Y_c and write:

$$\ddot{y} = \omega^2(y_c - y) - 2\zeta\omega\dot{y} \tag{16}$$

y and $\dot{y}$ are measurable while y_c is commanded. We can connect equation 15 to 11 by looking at equation 7. We find that the connection is $\theta_c = -\frac{\ddot{x}}{g}$. For equation 12 and 16, the connection is $\phi_c = \frac{\ddot{y}}{g}$. An integral was added to equations 11, 12, and 14. Equation 14 also had a gravity correction term of $\frac{mg}{cos(\theta)cos(\phi)}$ added in. For the setup, $\zeta = 0.7$, $\omega_{in} = 4$, and $\omega_{out} = 1$ were used in these equations.

With these equations, the quadcopter's controller is ready to receive trajectories from the mission planner of the quadcopter. Each motor can produce up to 10 N of force each. To protect the motor, I will include a saturation block that only allows the input to go to its max.

Added Notes: Previous to this design, I tried to use LMIs to solve the Lyapunov equation for an optimal controller. However, with this design I struggled with trajectory management, and decided to switch to a dynamic inversion controller. This way, I would have control over the x, y, and z coordinates. Dynamic inversion worked extremely well until I attempted to solve the multiple agent problem with it as my main controller. It turns out that dynamic inversion is extremely sensitive to unmodeled dynamics in the system. I tried to "cheat" and take "measurements" of the solved for force tension in the string (as described in III.B), and add them into the control law. However, the dynamics of the linked quadcopters would still "blow up". Consequently, this forced me o go investigate more robust controllers.

I started investigating creating a robust controller with Matlab's hinfsyn() and choosing appropriate weights. The problem I ran into was that all of the poles of the quadcopter are at 0 on the complex plane. This resulted in the robust controller solver being unable find a controller that could properly minimize:

$$\left\| \begin{matrix} W_s S \\ W_t T \end{matrix} \right\|_\infty$$

Since hinfsyn() did not work, I began reducing each subsystem to a SISO system so that I could use mixsyn() to create a robust controller through loop-shaping. I chose $\dot{x}$, $\dot{y}$, and $\dot{z}$ as the variables I wanted to control. The problem I came across with designing this controller was that it was *extremely* hard to attenuate the disturbance rejection (which is what would handle the unmodeled dynamics of the multi-agent problem). I did manage to design a controller on the linear side, but the nonlinear model "blew up" when the controller was applied. This is about where I ran out of time to keep investigating new controllers for the quadcopter.

If I had had more time, I would have looked into how to better integrate the mass dynamics into the controller, or model it as uncertainty in the plant. While not ideal, it might be the only way the quadcopter can handle the loaded mass. One could theoretically add a switch in the controller for when the quadcopter is with and without the weight. The next two things I would have liked to investigate is nonlinear control, and robust optimal control. I am aware of the existence of robust optimal control techniques, but do not know how to immediately implement them. Nonlinear control is probably the field I am least familiar with, but it might offer some useful options. When I linearized the system, I was surprised to find what the system reduced down to. I was expecting a much fuller A matrix after linearizing the system from past experience. It makes sense the quadcopter's A matrix looks as it does for trim position, but I felt like a lot of information on the dynamics was lost.

III. Multi-Quadcopter Problem Formulation

A. Task

This paper focuses on creating a decentralized mission planner in each quadcopter to perform a cooperative task. The initial design of the task will be three 1 kg quadcopters that have to lift a 0.9 kg mass, and maneuver it through an obstacle. The simulations section of this paper will better depict how the obstacle was setup for the quadcopter in the model. The following assumptions were made:

- The agent will have full access to the state space vectors of other agents as perceived (including the errors from sensors).

- Agents know the policies of other agents to speed the convergence of the critic since agents are performing a cooperative task.

B. Load Kinematics

We will need to slightly adjust the kinematics derived earlier to account for the three different quadcopters carrying a distributed load amongst them. We will first start with defining the load's kinematic equations in the inertial frame. The first step is to draw the vectors from the inertial frame to the quadcopter, and then from the quadcopter to the mass. The diagram below displays the vectors used:

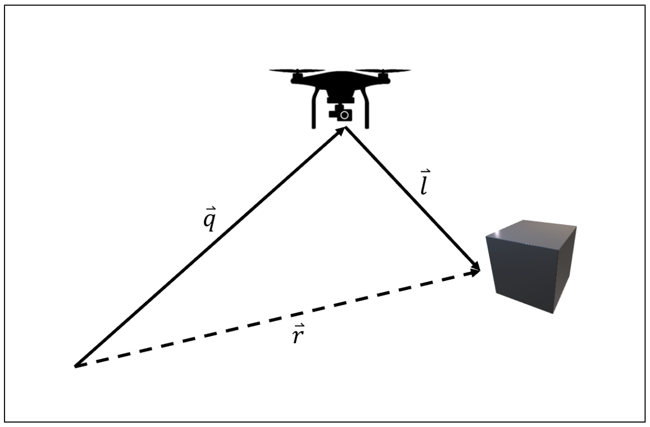

Fig. 5. The vectors of the quadcopter and mass system.

If we use $\frac{{}^{\mathcal{N}}d^2\vec{r}}{dt^2}$, we can capture the equations of motions along $\vec{r}$:

$$m_l \frac{{}^{\mathcal{N}}d^2\vec{r}}{dt^2} = \begin{bmatrix} 0 \\ 0 \\ m_l g \end{bmatrix}^{\mathcal{N}} - \vec{F_{l_1}} - \vec{F_{l_2}} - \vec{F_{l_3}} \tag{17}$$

The forces acting on the mass are the three tensions in the strings from the quadcopters and then gravity. Since the strings are attached to the center of the mass, the mass does not rotate and thus there is no need for a rotation matrix.

In equation 17, we have two knowns and three unknowns. We can move the mass over to the left so that the knowns are the on the same side. We know $\frac{{}^{\mathcal{N}}d^2\vec{r}}{dt^2}$ since we can use trilateration to solve for the intersection of the three spheres attached to the centers of the quadcopters to find the location of the mass. Code from [17] was used to calculate the intersection point. From there, we can take the double derivative to obtain $\frac{{}^{\mathcal{N}}d^2\vec{r}}{dt^2}$ from the location. After we have the location of the mass we can use the equation $\vec{l} = \vec{x}_m - \vec{x}_q$ to solve for the vector $\vec{l}$. Then $\hat{l} = \frac{l}{||l||_2}$. With this in mind, we can rearrange equation 17 to get:

$$m_l \frac{{}^{\mathcal{N}}d^2\vec{r}}{dt^2} - \begin{bmatrix} 0 \\ 0 \\ m_l g \end{bmatrix}^{\mathcal{N}} = \begin{bmatrix} \hat{l}_{1x} & \hat{l}_{2x} & \hat{l}_{3x} \\ \hat{l}_{1y} & \hat{l}_{2y} & \hat{l}_{3y} \\ \hat{l}_{1z} & \hat{l}_{2z} & \hat{l}_{3z} \end{bmatrix} \begin{bmatrix} \gamma_1 \\ \gamma_2 \\ \gamma_3 \end{bmatrix} \tag{18}$$

γ_i is the tension in each individual string, and can be solved for by performing some linear algebra. Once we have the γ vector, we can use the following equations to find all of the force tensions:

$$\vec{F_{l_1}} = \begin{bmatrix} F_{x1} \\ F_{y1} \\ F_{z1} \end{bmatrix} = \gamma_1 \begin{bmatrix} \hat{l}_{1x} \\ \hat{l}_{1y} \\ \hat{l}_{1z} \end{bmatrix} \tag{19}$$

$$\vec{F_{l_2}} = \begin{bmatrix} F_{x2} \\ F_{y2} \\ F_{z2} \end{bmatrix} = \gamma_2 \begin{bmatrix} \hat{l}_{2x} \\ \hat{l}_{2y} \\ \hat{l}_{2z} \end{bmatrix} \tag{20}$$

$$\vec{F_{l_3}} = \begin{bmatrix} F_{x3} \\ F_{y3} \\ F_{z3} \end{bmatrix} = \gamma_3 \begin{bmatrix} \hat{l}_{3x} \\ \hat{l}_{3y} \\ \hat{l}_{3z} \end{bmatrix} \tag{21}$$

We assume that the mass of the wire connecting the quadcopters to the load is negligible, and that the forces from the wire acts through the center of mass. This means we only need to adjust equation 2 for having the wire attached to it.

$$m \begin{bmatrix} \ddot{x} \\ \ddot{y} \\ \ddot{z} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ mg \end{bmatrix} + C_N^B \begin{bmatrix} 0 \\ 0 \\ -(F_1 + F_2 + F_3 + F_4) \end{bmatrix} + \vec{F_l} \tag{22}$$

C. Related Work

This paper draws most of its influence from three main sources: "Scalable Cooperative Transport of Cable-Suspended Loads with UAVs using Distributed Trajectory Optimization" [5], "Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments" [15], and "R-MADDGP for Partially Observable Environments and Limited Communication" [16]. The result is a paper that mixes different concepts and provides a layout of how to implement into Simulink.

Jackson, Howell, etc. in their paper used a quadcopter of similar specifications to carry a 0.9 kg load distributed by different agents. Their method proved successful, however, they used a constrained trajectory optimization solver to drive the mission planning of the agents [5]. In this paper, I propose, to use a technique grounded in Reinforcement Learning (RL). Due to time and money constraints of this project, this paper will stop at modeling the dynamics instead of continuing onto flight test like [5].

"Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments" will be the paper that provides much of the mathematical backbone for the mission planner. They took "Continuous Control with Deep Reinforcement Learning"'s algorithm for Deep Deterministic Policy Gradient (DDPG), and adjusted it to work for Multi-Agent Networks [15, 20]. The new algorithm Lowe, Wu, etc. created is called Multi-Agent Deep Deterministic Policy Gradient (MADDPG).

The last paper provides an extension of the previous paper by designing an algorithm called Recurrent Multi-Agent Deep Deterministic Policy Gradient (R-MADDPG). In this algorithm they train two different policies for navigation and communication- making it more robust to real world conditions such as partial observations [16]. Due to the time constraints of the course, I decided to narrow the scope and leave this algorithm adjustment out. For future work, I would start by adding the second policy for communication to the architecture to see how the performance changes.

Overall, this paper will differentiate itself in the following way:

1) The MADDPG algorithm will have to work within the constraints of a quadcopter and not throw it into instability.
2) The MADDPG algorithm will be used to solve a three dimensional problem vs a two dimensional problem.
3) The noise of the sensors in the quadcopter will be accounted for
4) The MADDPG algorithm will operate as the mission planner for the quadcopter as opposed to the controller of the agent.

D. *Multi-Agent Deep Deterministic Policy Gradient [15]*

Traditional Reinforcement Learning (RL) algorithms suffer from either an inherent non-stationary environment, or from variance as the number of agents is increased. Most work in the field of RL, however, has primarily been focused on the problem of a single agent. These algorithms require adjustments from the traditional algorithms to model or predict other agents' behaviors.

Multi-Agent Deep Deterministic Policy Gradient (MADDPG) has a few notable characteristics: it uses its own observations to make a decision, does not assume a differential model of the environment dynamics or any particular structure on the communication between agents, policy is continuous, and is an off policy algorithm. We want the agent to be autonomous in the sense that it looks around at its surroundings, and makes a decision based on what it receives. We also don't want the environment to be assumed since it could potentially be changing. An off-policy algorithm allows for continuous learning and exploration.

MADDGP is setup to be put through centralized training, but execute in a decentralized manner. Each agent contains its own policy (actor), while a critic is augmented with extra information about the policies of the other agents. The critic is the part of the algorithm that is put through centralized training, while the actor is the policy that is used during the execution phase to act in a decentralized manner. By having the critic be aware of the other agents' policies, the agent knows the actions of the other agents, and thus the environment can be perceived as stationary.

MADDPG is a RL algorithm, so that means it will have a reward structure to teach the agent the correct actions given the state of the mass and of itself. For example, crashing into a wall will give a large negative reward while achieving the objective will give a large positive reward. There will also be negative rewards if the algorithm provides outputs that are outside of the constraints of the agent- such that the agent might be thrown into instability. The details of the reward structure will be given in the Simulation Setup section of the paper.

Let N be the number of agents. The policies of the agents are parameterized by $\theta = \theta_1, ..., \theta_N$ and $\mu = \mu_1, ..., \mu_N$ is the set of continuous and deterministic agents' policies. $Q_i^\mu(x.a_1, ..., a_N)$ is the centralized action-value function that outputs the Q-value for agent i given the action of all agents $(a_1, ..., a_N)$. x is the state information of the agent. μ_{θ_i} can be abbreviated to μ_i:

$$\nabla_{\theta_i} J(\mu_i) = \\ \mathbb{E}_{x,a\sim\mathcal{D}}[\nabla_{\theta_i}\mu_i(a_i|o_i)\nabla_{a_i}Q_i^\mu(x.a_1, ..., a_N)|_{a_i=\mu_i(o_i)}]$$

The centralized action-value function Q_i^μ is updated by minimizing the loss:

$$\mathcal{L}(\theta_i) = \mathbb{E}_{x,a,rx'}[(Q_i^\mu(x, a_1, ..., a_N) - y)^2]$$

$$y = r_i + \gamma Q_i^{\mu'}(x', a_1', ..., a_N')|_{a_j'=\mu_j'(o_j)}$$

Here, $\mu' = \mu_{\theta_1'}, ..., \mu_{\theta_N'}$ is the set of target policies with delayed parameters θ_i'.

On the next page is the algorithm used to perform MADDPG as given in [15]. μ and Q^μ will be saved as neural networks in the algorithm. $\mathcal{L}$ is the equation used to adjust the weights in the neural network for Q, and will be computed through gradient descent. $\nabla_\theta J$ adjusts is weights for μ through gradient ascent to find a maximum instead of a minimum.

Algorithm 1 MADDPG Algorithm

```
for episode = 1 to M do
    Initialize a random process $\mathcal{N}$ for action exploration
    Receive initial state $x$
    for $t = 1$ to max-episode-length do
        for each agent $i$, select action $a_i = \mu_{\theta_i}(o_i) + \mathcal{N}_t$ w.r.t the current policy and exploration
        Execute actions $a = (a_1, ..., a_N)$ and observe reward $r$ and new state $x'$
        Store $(x, a, r, x')$ in replay buffer $\mathcal{D}$
        $x \leftarrow x'$
        for agent $i = 1$ to N do
            Sample a random minibatch of $S$ samples $x^j, a^j, r^j, x'^j$ from $\mathcal{D}$
            Set:
            $y^j = r_i^j + \gamma Q_i^{\mu'}(x'^j, a'_1, ..., a'_N) \mid_{a'_k = \mu'_k(o_k^j)}$
            Update critic by minimizing the loss:
            $\mathcal{L}(\theta_i) = \frac{1}{S}\sum_j (y^j - Q_i^{\mu}(x^j, a_1^j, ..., a_N^j))^2$
            Update the actor using the sampled policy gradient:
            $\nabla_{\theta_i} J \approx$
            $\frac{1}{S}\sum_j \nabla_{\theta_i}\mu_i(o_i^j)\nabla_{a_i} Q_i^{\mu}(x^j, a^j, ..., a_N^j) \mid_{a_i = \mu_i(o_i^j)}$
        end for
        Update target network parameters for each agent $i$:
        $\theta'_i \leftarrow \tau\theta_i + (1 - \tau)\theta'_i$
    end for
end for
```

E. Algorithm Structure

I will break the quadcopter architecture into two different parts- the controller and then the mission planner. The controller was talked about in depth in the Controller Design section under Quadcopter Design, and the mission planner algorithm was talked about in the Multi-Agent Deep Deterministic Policy Gradient section. In this section, we will talk about combining the mission planner and the controller.

The mission planner utilizes the MADDPG algorithm to define coordinates in the x, y, and z space for the agent to fly to. These coordinates are then fed to the controller through the reference signal. During training, the MADDPG algorithm will receive the quadcopter's state, the mass's state, and the other quadcopters' states as inputs. This allows for the critic to be trained with extra information to help train the policy, μ. After training, each quadcopter will have its own deterministic policy that will provide the agent x, y, and z coordinates in the Earth frame for the agent to go fly to. Below is a picture of the current algorithm structure:

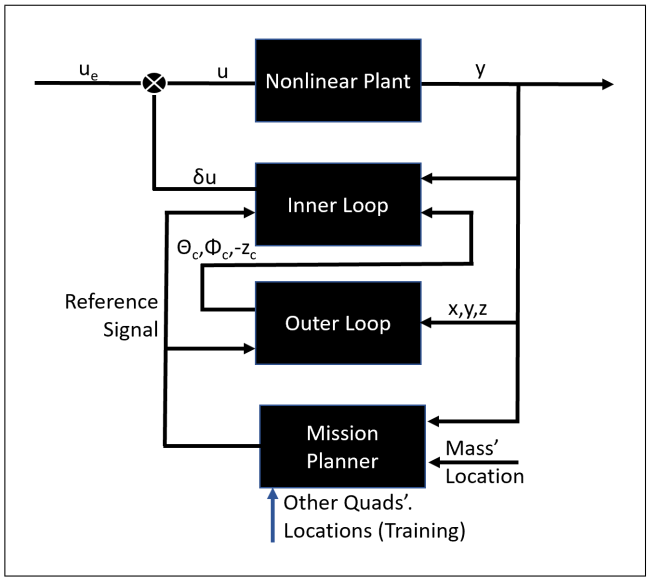

Fig. 6. The controller connected to the mission planner.

IV. Simulation Setup

A. Operating Space of Quadcopters

For the simulation setup, I will define a $(6, 6, 6)$ m space. In this space I will create a deadzone that kills the agent if they were to "crash" into it. There is a doorway for which the agents will need to carry the load through the deadzone that will take the shape of a $(1.6, 2.1, 1)$ m tunnel. Below is a picture of the space defined within Matlab.

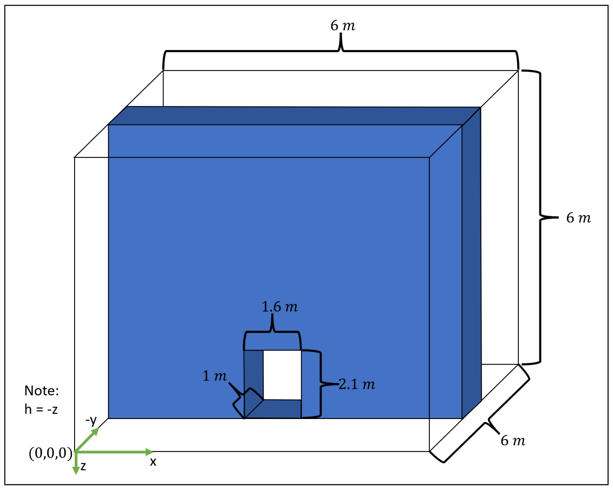

Fig. 7. Tunnel for quadcopters to maneuver through.

The quadcopters and mass will be initialized to the following locations:

$$x_{1_0} = \begin{bmatrix} 3 \\ 1 \\ -(1+\sqrt{2}) \end{bmatrix}$$

$$x_{2_0} = \begin{bmatrix} 2 \\ 1.5 \\ -(1+\sqrt{1.25}) \end{bmatrix}$$

$$x_{3_0} = \begin{bmatrix} 4 \\ 1.5 \\ -1+\sqrt{1.25} \end{bmatrix}$$

$$x_{m_0} = \begin{bmatrix} 3 \\ 1.5 \\ -1 \end{bmatrix}$$

B. *Reward Structure*

In the rewards structure for the quadcopters, there will be four main categories as summarized in table II. The crash award will occur anytime the quadcopter or mass crashes. A quadcopter might crash if it runs into the wall, hits another quadcopter, or is thrown into instability by the mission planner's fly to point. The quadcopter will be penalized for leaving the $6\times6\times6$ m space. The objective is met when the quadcopters maneuver the object to the other side of the wall successfully. When the mass crashes or the objective is met, all agents will receive the reward. To prevent the quadcopters from being content with not crossing, there is a -1 reward for every action they take. For example, if the quadcopter decides to do nothing for an action, it will still receive a -1 reward.

TABLE II
REWARD STRUCTURE

Type	Value
Crash	-100
Outside Boundaries	-100
Objective Met	+1000
Actions	-1

C. *Matlab DDPG Agent Set-Up*

This project attempted to keep everything in-house of Matlab and Simulink since it is the industry standard for GNC in aerospace engineering. This section quickly breaks down how the agent was defined with the MAD-DGP algorithm.

The mission planner takes the inputs of GPS, 6 DoF states, and mass location. The mass' location and GPS signal build the observation vector- which is the difference between the quadcopters and mass' location. Then there was a reward function that takes the location of all the quadcopters and the mass and outputs both a reward and isdone flag. Simulink has a block that contains the agent called RL agent. The agent object input for the block gets defined through a Matlab script. This Simulink block outputs a size three vector that contains x and y coordinates, and the altitude, h, as well.

The agent script on the Matlab side defines the size of the observations and actions that the agent can take and output. From there, some options are defined for the agent before creating the DDPG Agent. Three identical quadcopter agents are created, stacked into a vector, and then used to create an environment with the Simulink model. validateEnvironment(env) checks that everything within the Simulink model is working before attempting to start the training of the agents.

V. MAIN RESULTS

A. *Results*

The Matlab and Simulink files can be found in the following Github repository: https://github.com/tori1323/Multi-Agent_Control. To open the Simulink files, the user must be using Matlab 2022a. Matlab added a lot of features to its reinforcement learning toolbox in the 2022 release that do not exist in the previous packages. This can lead to confusion when looking through Matlab's documentation for the RL toolbox.

These files will provide a baseline for which someone can proceed with to eventually solve the three agents carrying a load problem. Some notes on the existing model, and how I would move forward with training if the controller is fixed:

- Train a mission planner without the quadcopter model first. One of the things that occurred to me while trying find an appropriate controller was that the model would never train if the quadcopter kept falling out of the sky. Simulink usually defaults to the ode45 solver- which uses a variable step size. When the model is acting in sporadic manner (such as the quadcopter falling out the sky), the ode algorithm will gradually reduce its step size in an attempt to solve the problem. However, the algorithm ends up getting stuck for a long time before crashing when this happens. Consequently, the mission planner needs some training beforehand so that no crazy coordinates are outputted- causing the quadcopter model to tumble all the time. The mission planner will then need to go through some more training so that outputs are truly optimized for the problem.
- I would first train the model without noise to see how the agents perform with that baseline.
- Noise and saturation blocks need to be added to the existing models. I did not want to add the saturation blocks to protect the motors until I saw how control inputs performed without interruption. 10 N was a

rough requirement I made after observing the performance of a single quadcopter tracking specified coordinates.

VI. Conclusions

A. Contributions

This paper provides a basic model that could be grown upon in further work with quadcopter mission planning with the MADDGP algorithm. This paper provides detailed accounts of how everything was derived, problems faced, and where further work can start. The files are all saved in the Github repository: https://github.com/tori1323/Multi-Agent_Control. This paper also shows how one could possibly use the recent additions to the reinforcement learning package in Matlab to utilize the MADDGP algorithm within the Simulink environment.

One of the last contributions of this paper was to introduce the idea of using the MADDGP algorithm as a mission planner instead of an actual controller. This paper very clearly differentiates the structures of the controller through "classical" control techniques, and then using reinforcement learning algorithms to provide inputs into that controller.

B. Further Improvements

There were many assumptions that were made through out the paper to simplify the task. Some of the first things I would do to increase the fidelity of the model are:

- Collect a CAD model of the quadcopter with all of the payloads attached. The CAD model should give an inertial matrix with non-zero products.
- Model the motors of the quadcopter. For the sake of time, I left the control inputs simplify as forces. In reality, modeling the dynamics of the motors would greatly effect the outcome of the system.
- Model disturbances to the system such as wind. This would be easier to do if there was a CAD model of the quadcopter. Small UAVs, in general, struggle with flying near buildings due to the gusts of wind that come bouncing off the walls. These quadcopters could potentially be operating in places with buildings, so it would be good to better model the environment.

Last, the VMS system should be loaded into actual mechanical quadcopters, and be validated through test. All models are incorrect by nature, so the only way to truly validate how effective the model is through test.

The following features would be great add-ons in future work:

- Add either a camera or other sensor for the quadcopter to detect surrounding. In this paper, I had the agent learn the stationary environment through GPS location, but it would be more useful for the quadcopter to detect its environment and adjust accordingly.
- Add a second policy for communication between the agents. In reality, the agents will not be able to continuously talk to each-other.
- Detail reward structure for specific states of the quadcopter. For example, if the way-point causes the quadcopter to tilt more than 40 degrees in one direction, penalize it for giving such a point. This would require doing a more detailed analysis of the quadcopters' flying performance qualities.
- Increase the amount of scenarios that the quadcopters are trained through. I designed one scenario due to the time constraint- which means the algorithm will probably be trained to be "brittle" to change.
- Use other languages like Python to train the neural networks. I wanted to exclusively use Matlab/Simulink to see how far I could go with using just one platform. One of the most disappointing things about using Simulink was that the actor had to use the same observations that the critic received. In future work, it would make sense for the critic to receive more information than what the actor receives. Simulink, unfortunately, was a little limiting in this factor.
- A more robust controller than can reject the unmodeled dynamics created from being attached to the mass that is attached to two other quadcopters.
- Test how well the mission planner works with controlling the speeds of the quadcopter instead of positions. The speeds of the quadcopter are a lot less intuitive to control- which is why I was trying to keep to positions in this project.

References

[1] Jack Daniels, "What Can Quadcopters Be Used For", *Systems Control Letters*, 1 April 2020.

[2] Science and Technology Directorate, "Feature Article: As Drone Popularity and Potential Risk Soars, so too does ST Preparedness", https://www.dhs.gov/science-and-technology/news/2021/07/16/feature-article-drone-popularity-and-potential-risk-soars-st-prep

[3] ImproDrone, "Why is the Quadcopter a Popular Design for Smaller Drones?", https://improdrone.com/why-is-the-quadcopter-a-popular-design-for-smaller-drones/.

[4] Business Insider, "Drone technology uses and applications for commercial, industrial and military drones in 2021 and the future", *Insider Intelligence*, 12 January, 2021.

[5] Brian E. Jackson, Taylor A. Howell, Kunal Shah, Mac Schwager, and Zachary Manchester, "Scalable Cooperative Transport of Cable-Suspended Loads With UAVs Using Distributed Trajectory Optimization", *IEEE ROBOTICS AND AUTOMATION LETTERS*, Vol. 5, No. 2, April 2020.

[6] Miriam McNabb, "Flying a Drone in the Wind, Cold, and Other Challenging Environments", *Drone Life*, 28 June 2021.

[7] Robert Nelson, "Flight Stability and Automatic Control", *McGraw-Hill Education*, 2nd Edition, 1 October 1997.

[8] PRG UMD Teaching, "Class 6 - Quadrotor Dynamics", https://www.youtube.com/watch?v=UC8W3SfKGmg.

[9] Wilselby, "Quadrotor System Modeling - Nonlinear Equations of Motion", *Systems Control Letters*, 1 April 2020.https://www.wilselby.com/research/arducopter/modeling/.

[10] Richard Fitzpatrick, "Moment of Inertia Tensor".

[11] Ana Hoverbear, "Quadcopters: Sensors", 2nd Edition, 6 June 2015.

[12] Chris Winkler, "How Many Sensors are in a Drone, And What do they Do?", *Fierce Electronics*, 21 July 2016.

[13] Drone Frame and Accessories, "F330 Glass Fiber Mini Quadcopter Frame 330mm", *Robu.in*.

[14] Silas Kevin Isaac Graham, "Development of a Quadrotor Slung Payload System", *University of Toronto*, 2019.

[15] Ryan Lowe, Yi Wu, Aviv Tamar, Jean Harb, Pieter Abbeel, and Igor Mordatch, "Mutli-Agent Actor-Critic for Mixed Cooperative-Competitive Environments", *arXiv*, 14 Mar 2020.

[16] Rose E. Wang, Michael Everett, and Jonathan P. How, "R-MADDPG for Partially Observable Environments and Limited Communication", *arXiv*, 18 Feb 2020.

[17] Andrew Wagner, https://stackoverflow.com/questions/1406375/finding-intersection-points-between-3-spheres, *StackOverflow*.