Model Predictive Control for Cooperative Hunting in 3D Environment

ESC499Y1 2019-2020 Final Presentation

Student Name: Che (Charles) Liu

Student Number: 1002246839

Supervisor: Prof. Hugh H.T. Liu

Division of Engineering Science, University of Toronto

March 27, 2020

Background

- Cooperative hunting is a popular area of study in robotics
- It is a collective behavior commonly seen in nature, especially in pack hunters or social predators
  - Wolfs, wild dogs, falcons, chimpanzees, etc.
- More likely to capture the prey and prevents it from escaping

Figure 1: Mollies pack wolves hunting a bison.

Why three-dimensional?

- ❑Most of existing solutions only consider the 2D case or collapse the 3D problem into 2D
- ❑The UAV dynamics can be better modelled in 3D, which allows for more optimal surroundings

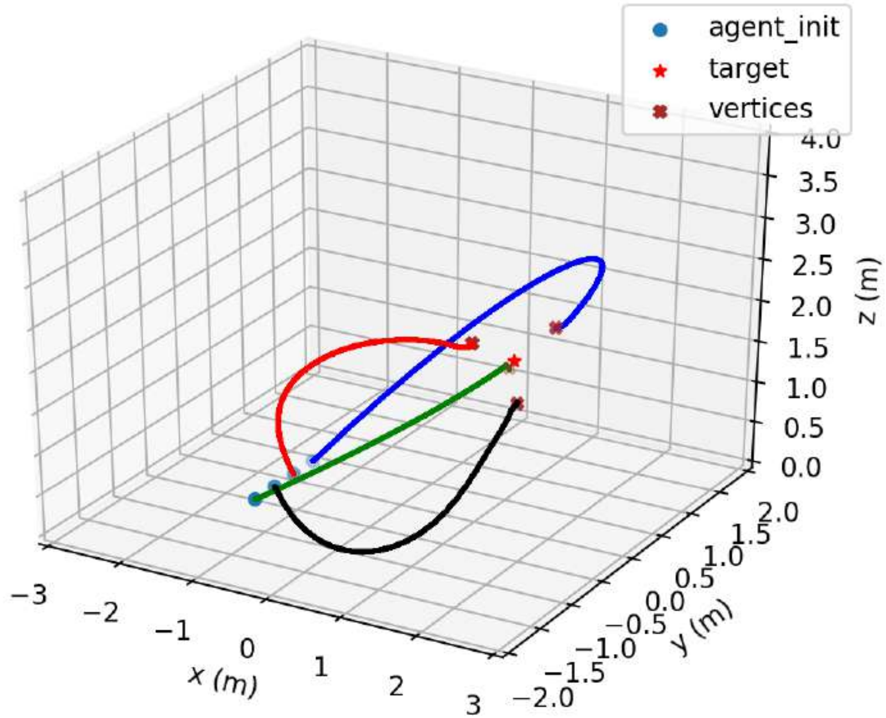


Figure 2: An example of cooperative hunting in 3D.

Crazyflie 2.1 + Robot Operating System (ROS)

- A flying development platform based on a nano-quadcopter
  - Lightweight
  - Small in size
  - Flexible
  - Robot Operating System(ROS)
- Ideal for hunting purposes
- Takes the desired [roll, pitch, yaw rate, thrust] through *cmd_vel*

Figure 3: A Crazyflie 2.1 nano-quadcopter.

Figure 4: Robot Operating System (ROS).

Design Overview

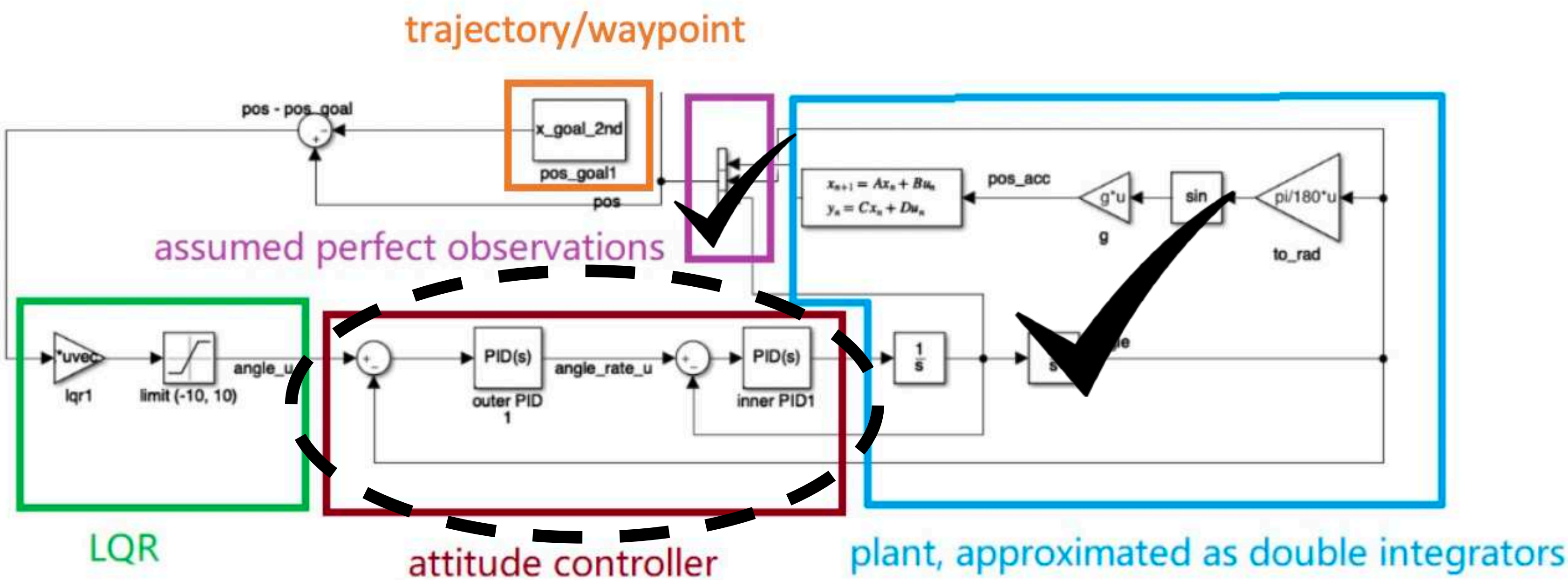


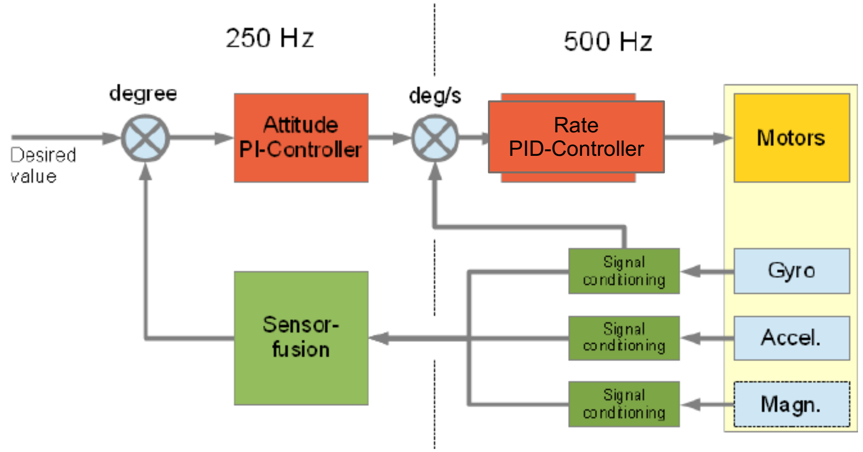
Figure 5: Different modules in the thesis design.

- ❑ Low-level to high-level design

1. Crazyflie 2.1 attitude controller approximation
2. Linear quadratic regulator (LQR) for roll and pitch
3. Trajectory controller design

Attitude Controller Approximation

- ❑Crazyflie uses a cascade PID for attitude control
- ❑Abstract the motors away and approximate the plant as a double integrator
- ❑Pulse width modulation (PWM) → acceleration



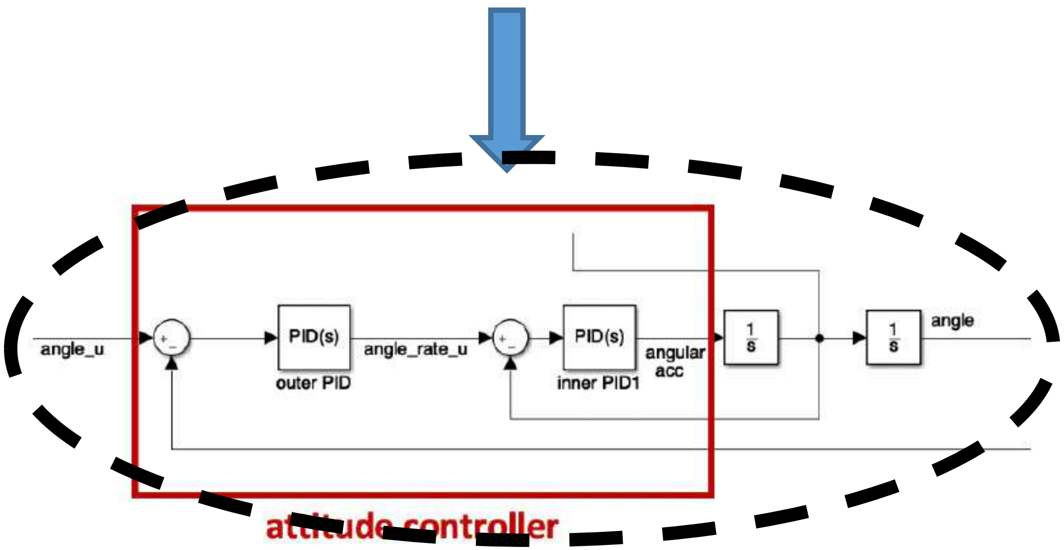


Figure 6: Attitude controller approximation.

Attitude Controller Approximation

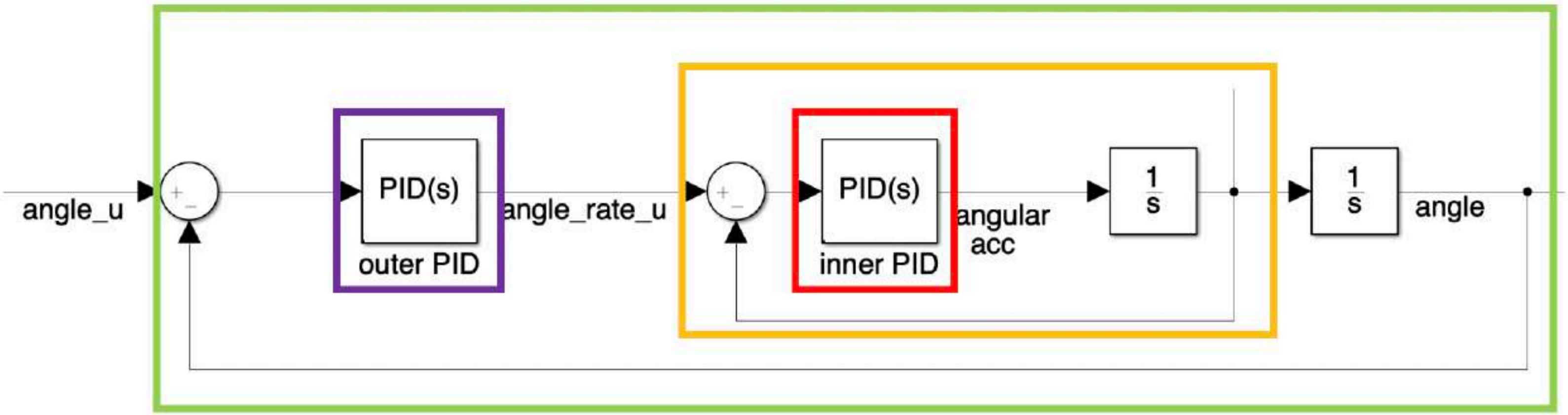


Figure 7: Block diagram of the approximated block diagram.

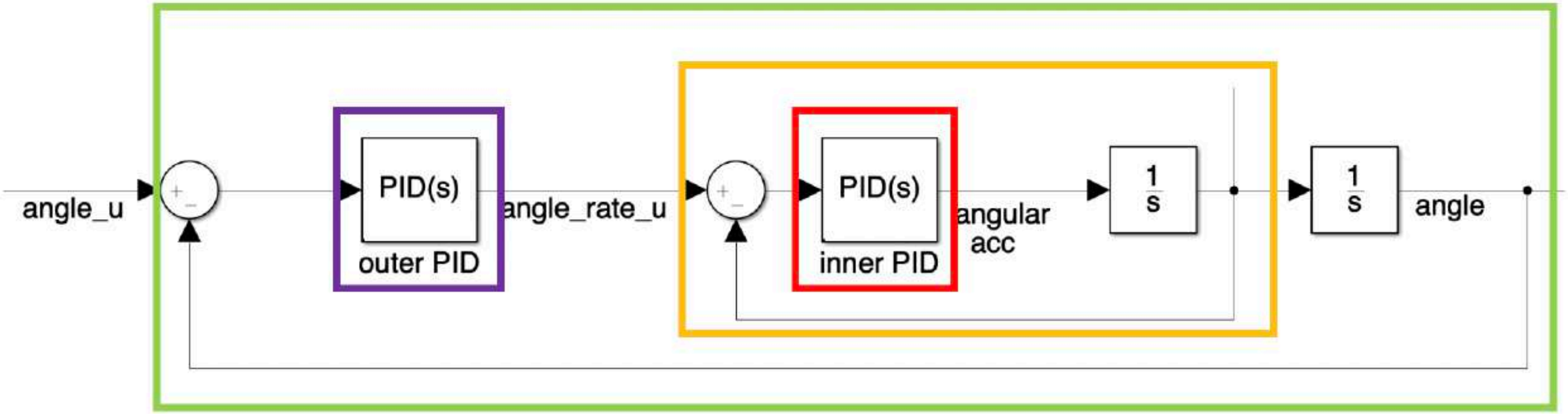
Red → 'inner PID'

Purple → 'outer PID'

Orange → 'inner closed-loop system'

Green → 'outer closed-loop system'

Attitude Controller Approximation



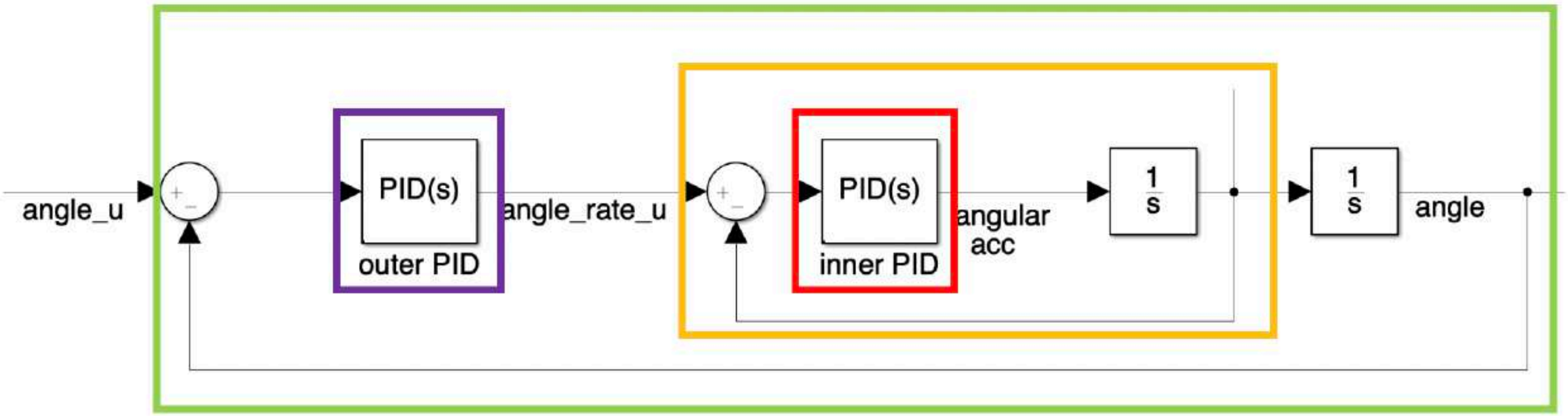
- The gains used for the cascade PID controller are:

$$k_{p_{inner}} = 250.0, k_{i_{inner}} = 500.0, k_{d_{inner}} = 2.5, ilim_{inner} = 33.0$$

$$k_{p_{outer}} = 6.0, k_{i_{outer}} = 3.0, k_{d_{outer}} = 0.0, ilim_{outer} = 20.0$$

$$G(s) = \frac{u(s)}{e(s)} = k_p + \frac{k_i}{s} + k_d s = \frac{k_d s^2 + k_p s + k_i}{s}$$

Attitude Controller Approximation



Substitute the gains in, and after some manipulations

$$G(s)_{inner} = \frac{2.5s^2 + 250s + 500}{s}$$

$$G(s)_{outer} = \frac{6s + 3}{s}$$

$$G(s)_{innerCLS} = \frac{G(s)_{inner}\frac{1}{s}}{1 + G(s)_{inner}\frac{1}{s}} = \frac{0.7143s^2 + 71.43s + 142.9}{s^2 + 71.43s + 142.9}$$

$$\boxed{G(s)_{outerCLS} = \frac{G(s)_{outer}G(s)_{innerCLS}\frac{1}{s}}{1 + G(s)_{outer}G(s)_{innerCLS}\frac{1}{s}} = \frac{4.286s^3 + 430.7s^2 + 1071s + 428.6}{s^4 + 75.71s^3 + 573.6s^2 + 1071s + 428.6}}$$

Model Order Reduction

$$G(s)_{outer_{CLS}} = \frac{G(s)_{outer} G(s)_{innerCLS} \frac{1}{s}}{1 + G(s)_{outer} G(s)_{innerCLS} \frac{1}{s}} = \frac{4.286s^3 + 430.7s^2 + 1071s + 428.6}{s^4 + 75.71s^3 + 573.6s^2 + 1071s + 428.6}$$

- Transfer function of the cascade-PID controller
- 4th order transfer function
  - Eight-dimensional state-space representation if we include both roll and pitch
  - High computational complexity
  - Prune to numerical issues
  - Difficult to interpret the states
- Can approximate the system with a lower-order one

Model Order Reduction

$$G(s)_{outer_{CLS}} = \frac{G(s)_{outer}G(s)_{innerCLS}\frac{1}{s}}{1+G(s)_{outer}G(s)_{innerCLS}\frac{1}{s}} = \frac{4.286s^3+430.7s^2+1071s+428.6}{s^4+75.71s^3+573.6s^2+1071s+428.6}$$

- MATLAB's *[sysb, g] = balreal(sys)* function computes a balanced realization of *sysb*
  - *g* contains the Hankel singular values for each state
  - Small value indicates that the corresponding state has low energy and can be removed
  - For our system, the values are 0.5490, 0.0385, 0.0108, and 0.0004, which suggests that the first (and possibly the second) state should suffice
- MATLAB's *rsys = modred(sys, elim, method)* function computes the system after state elimination
  - First-order system reduction
  - Second-order system reduction

First-Order Order Reduction

- The transfer function becomes:

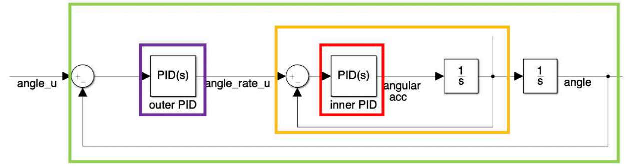
$$\widetilde{G}(s)_{CLS} = \frac{5.836}{s + 5.316}$$

- The corresponding state-space model in controllable canonical form is:

$$\mathbf{A} = -5.316,\ \mathbf{B} = -2.416,\ \mathbf{C} = -2.416,\ \mathbf{D} = 0$$

$$\dot{\boldsymbol{x}} = \mathbf{A}\boldsymbol{x} + \mathbf{B}\boldsymbol{u}$$

$$\boldsymbol{y} = \mathbf{C}\boldsymbol{x}$$



- To obtain the output dynamics, a coordinate transform from the internal state to the output need to be performed

$$\boldsymbol{y} = \mathbf{T}(\boldsymbol{x}) = \mathbf{P}^{-1}\boldsymbol{x} = \mathbf{C}\boldsymbol{x}$$

$$\hat{\mathbf{A}} = \mathbf{P}^{-1}\mathbf{A}\mathbf{P} = \mathbf{C}\mathbf{A}\mathbf{C}^{-1} = -5.3155$$

$$\hat{\mathbf{B}} = \mathbf{P}^{-1}\mathbf{B} = \mathbf{C}\mathbf{B} = 5.8363$$

$$\dot{\phi} = -5.3155\phi + 5.8363u$$

First-Order Order Reduction

- At steady state, the angle converges to:

$$\dot{\phi} = 0 = -5.3155\phi + 5.8363u$$

$$\phi = \frac{5.8363}{5.3155}u > u$$

- To get rid of the steady-state error, average the coefficient:

$$\dot{\phi} = -\frac{5.3155 + 5.8363}{2}\phi + \frac{5.3155 + 5.8363}{2}u = -5.5759\phi + 5.5759u$$

- Max diff in time delay:

$$(\Delta time\ delay)_{max} = \frac{(\Delta\phi)_{max}}{f}\frac{\pi}{180} \approx 2\ ms$$

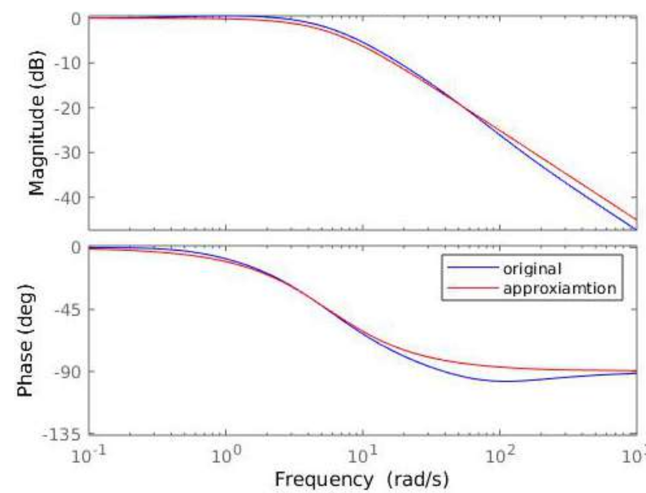


Figure 8: Bode plot of the reduced-order system vs. the original system

First-Order Order Reduction

- Step response
  - No steady-state error
  - Comparable rise time (~ 0.4 s)
  - Smaller overshoot

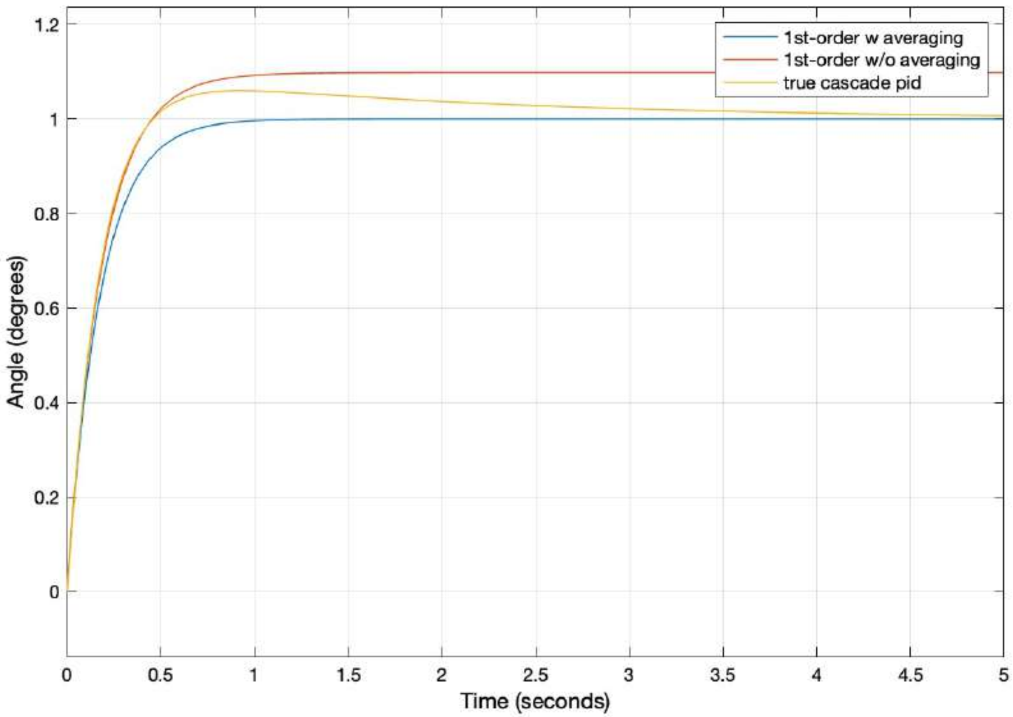


Figure 10: Step response of the first-order reduced system.

First-Order Order Reduction

- The state-space model with state $\boldsymbol{x} = (x, y, \dot{x}, \dot{y}, \phi, \theta)$ is thus:

Double Integrator

First-order Attitude

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \ddot{x} \\ \ddot{y} \\ \dot{\phi} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & g\frac{\pi}{180} \\ 0 & 0 & 0 & 0 & -g\frac{\pi}{180} & 0 \\ 0 & 0 & 0 & 0 & -5.5759 & 0 \\ 0 & 0 & 0 & 0 & 0 & -5.5759 \end{bmatrix} \begin{bmatrix} x \\ y \\ \dot{x} \\ \dot{y} \\ \phi \\ \theta \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 5.5759 & 0 \\ 0 & 5.5759 \end{bmatrix} \begin{bmatrix} u_\phi \\ u_\theta \end{bmatrix}$$

- where a small angle approximation $tan(\theta) \approx \theta$ (in radians) was used for accelerations in x and y directions

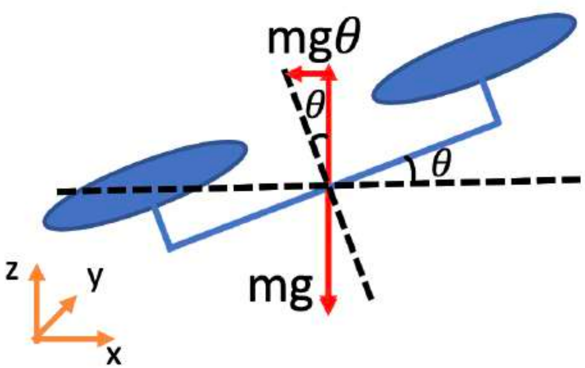


Figure 9: Force diagram of a quadcopter.

$$tan(\theta) = \frac{F_x}{mg}$$

$$\ddot{x} = \frac{F_x}{m} \approx g\theta(rad) = g\frac{\pi}{180}\theta(deg)$$

$$\ddot{y} = \frac{F_y}{m} \approx -g\phi(rad) = -g\frac{\pi}{180}\phi(deg)$$

Second-Order Order Reduction

- The transfer functions reduces to:

$$\widetilde{G}(s)_{angle} = \frac{angle}{angle_d} = \frac{4.321s + 417}{s^2 + 73.08s + 384.4}$$

$$\widetilde{G}(s)_{rate} = \frac{rate}{angle_d} = \frac{4.286s^2 + 419.4s - 12.89}{s^2 + 73.08s + 384.4}$$

- In controllable canonical form:

$$\mathbf{A} = \begin{bmatrix} -1.84 & -15.68 \\ 16.16 & -71.24 \end{bmatrix}, \quad \mathbf{B} = \begin{bmatrix} -3.337 \\ 10.81 \end{bmatrix}$$

$$\mathbf{C} = \begin{bmatrix} -1.031 & 0.08154 \\ 3.174 & 10.81 \end{bmatrix}, \quad \mathbf{D} = \begin{bmatrix} 0 \\ 4.286 \end{bmatrix}$$

- The rate transfer function is not strictly proper
  - we cannot use the same coordinate transform to get the dynamic in the output angle

Second-Order Order Reduction

- Can use the step response characteristics of the original system to design a second order system
- Rise time and percent overshoot from MATLAB's *stepinfo(sys)* function

$$T_r = 0.2948,\ M_p = 5.9900$$

- The undamped natural frequency and damping ratio can be calculated

$$\omega_n = \frac{1.8}{T_r} = 6.1058$$

$$\xi = \frac{|ln(M_p/100)|}{\sqrt{\pi^2 + ln^2(M_p/100)}} = -0.6673$$

$$G(s) = \frac{\omega_n^2}{s^2 + 2\xi\omega_n s + \omega_n^2} = \frac{37.2808}{s^2 + 8.1488s + 37.2808}$$

Second-Order Order Reduction

$$G(s) = \frac{\omega_n^2}{s^2 + 2\xi\omega_n s + \omega_n^2} = \frac{37.2808}{s^2 + 8.1488s + 37.2808}$$

- In controllable canonical form:

$$\mathbf{A} = \begin{bmatrix} -8.149 & -4.66 \\ 8 & 0 \end{bmatrix}, \mathbf{B} = \begin{bmatrix} 8 \\ 0 \end{bmatrix}$$

$$\mathbf{C} = \begin{bmatrix} 0 & 0.5825 \\ 4.66 & 0 \end{bmatrix}, \mathbf{D} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

- After the same coordinate transform:

$$\hat{\mathbf{A}} = \mathbf{CAC}^{-1} = \begin{bmatrix} 0 & 1 \\ -37.2808 & -8.1488 \end{bmatrix}$$

$$\hat{\mathbf{B}} = \mathbf{CB} = \begin{bmatrix} 0 \\ 37.2808 \end{bmatrix}$$

Second-Order Order Reduction

- Step response
  - Similar rise time to the first order system
  - Bigger overshoot than the first order system (~ 5%)

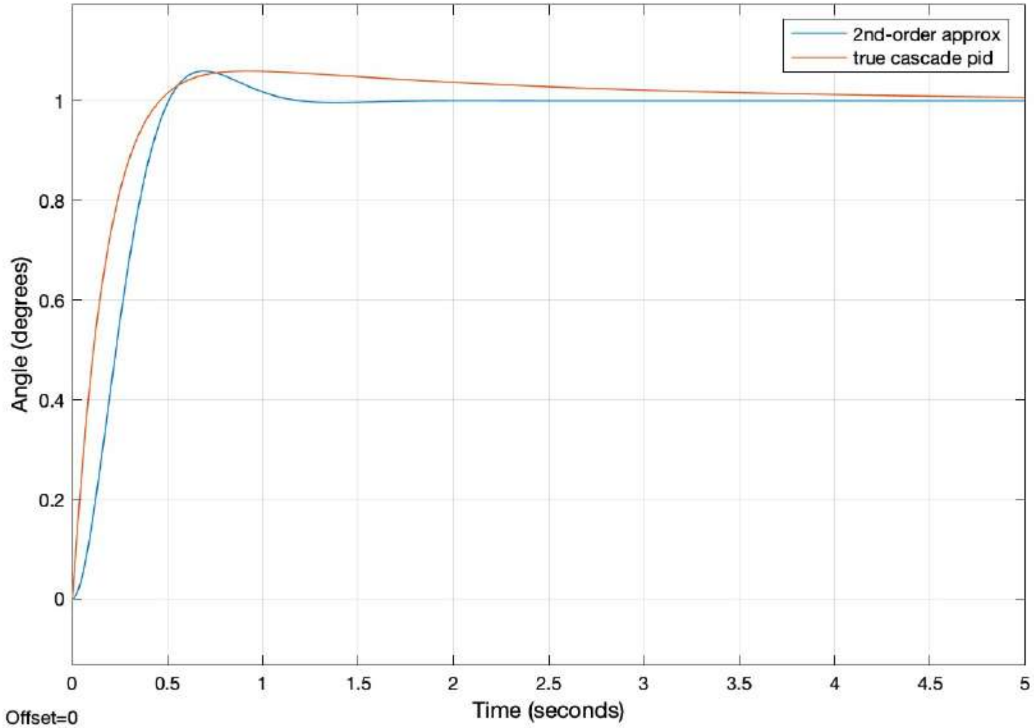


Figure 11: Step response of the second-order reduced system.

Second-Order Order Reduction

- The state-space model with state $\boldsymbol{x} = (x, y, \dot{x}, \dot{y}, \phi, \theta, \dot{\phi}, \dot{\theta})$ is thus:

Double Integrator

Second-order Attitude

$$
\begin{bmatrix} \dot{x} \\ \dot{y} \\ \ddot{x} \\ \ddot{y} \\ \dot{\phi} \\ \dot{\theta} \\ \ddot{\phi} \\ \ddot{\theta} \end{bmatrix}
=
\begin{bmatrix}
0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & g\frac{\pi}{180} & 0 & 0 \\
0 & 0 & 0 & 0 & -g\frac{\pi}{180} & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & -37.2808 & 0 & -8.1488 & 0 \\
0 & 0 & 0 & 0 & 0 & -37.2808 & 0 & -8.1488
\end{bmatrix}
\begin{bmatrix} x \\ y \\ \dot{x} \\ \dot{y} \\ \phi \\ \theta \\ \dot{\phi} \\ \dot{\theta} \end{bmatrix}
+
\begin{bmatrix}
0 & 0 \\
0 & 0 \\
0 & 0 \\
0 & 0 \\
0 & 0 \\
0 & 0 \\
37.2808 & 0 \\
0 & 37.2808
\end{bmatrix}
\begin{bmatrix} u_\phi \\ u_\theta \end{bmatrix}
$$

System Discretization

- The state-space model is in the continuous-time domain and needs to be discretized
- Since sampling time (~0.01 s) << time constant (~0.25 s), all discretization methods produce similar results
- Zero Order Hold (ZOH) method
- Each sample value is held constant for one sample interval

$$\mathbf{y}_k = \mathbf{y}\left(t_k\right), \quad t_k = kdt, k = 0, 1, 2, 3, \cdots$$

$$\boldsymbol{u}(t) = \boldsymbol{u}_{t_k}, \quad t_k \leq t < t_{k+1}, k = 0, 1, 2, 3, \cdots$$

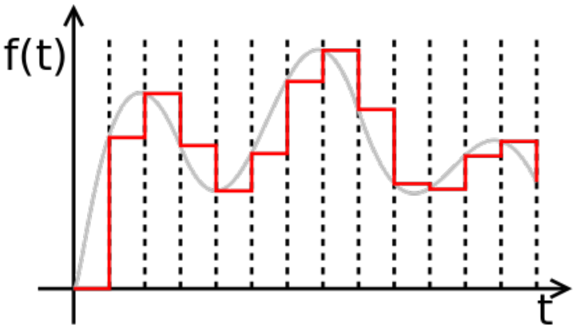


Figure 12: ZOH to discretize a signal.

System Discretization

- Using MATLAB's *sysd = c2d(sysc, dt)* function, the discretized system with *dt = 0.01s* is:

$$\boldsymbol{x}_{k+1} = \mathbf{A}_d \boldsymbol{x}_k + \mathbf{B}_d \boldsymbol{u}_k$$

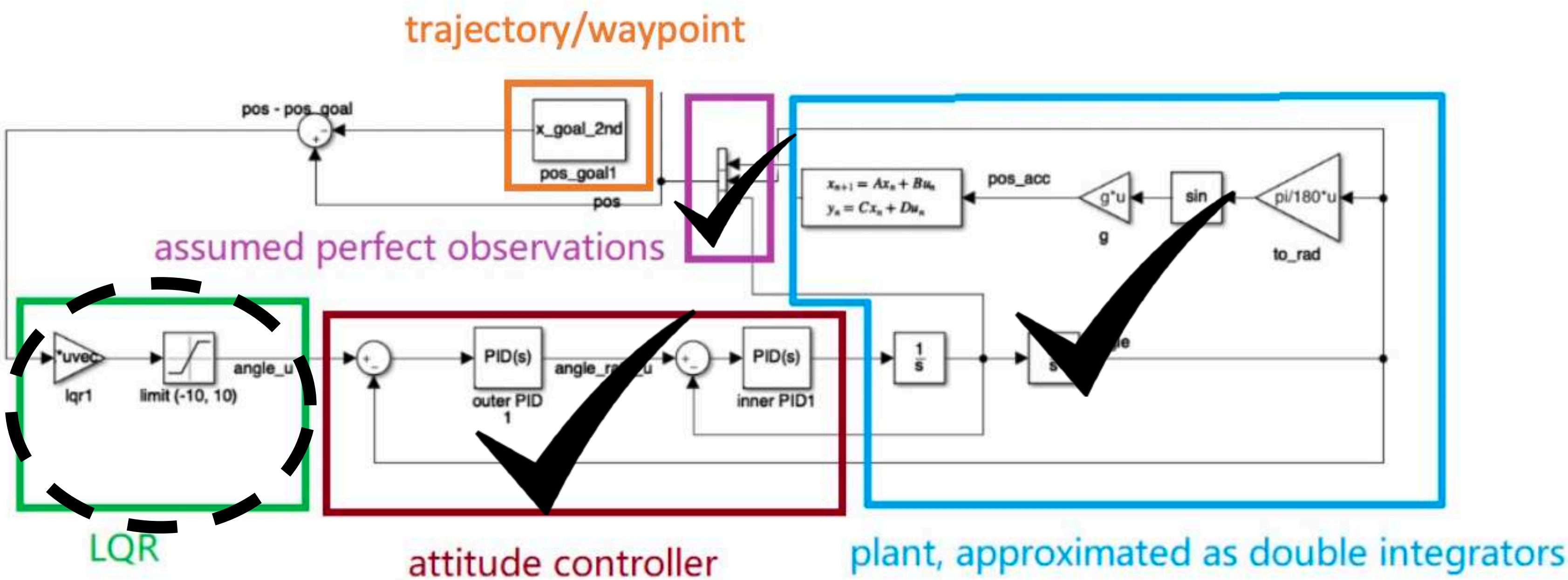
- For the first-order reduced system:

$$\mathbf{A}_d = \begin{bmatrix} 1 & 0 & 0.01 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0.01 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0.001665 \\ 0 & 0 & 0 & 1 & -0.001665 & 0 \\ 0 & 0 & 0 & 0 & 0.9458 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.9458 \end{bmatrix}, \mathbf{B}_d = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0.05423 & 0 \\ 0 & 0.05423 \end{bmatrix}$$

System Discretization

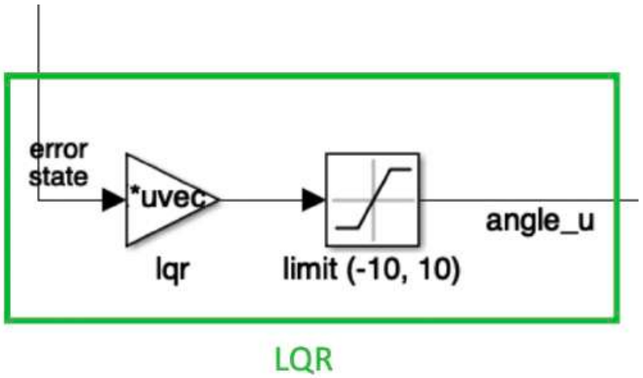
- For the second-order reduced system:

$$\mathbf{A}_d = \begin{bmatrix} 1 & 0 & 0.01 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0.01 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0.001711 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0.001711 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.9982 & 0 & 0.009597 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.9982 & 0 & 0.009597 \\ 0 & 0 & 0 & 0 & -0.3578 & 0 & 0.92 & 0 \\ 0 & 0 & 0 & 0 & 0 & -0.3578 & 0 & 0.92 \end{bmatrix}, \mathbf{B}_d = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0.001814 & 0 \\ 0 & 0.001814 \\ 0.3578 & 0 \\ 0 & 0.3578 \end{bmatrix}$$



trajectory/waypoint
pos - pos_goal
x_goal_2nd
pos_goal1
pos
pos_acc
sin
g
to_rad
assumed perfect observations
angle_u
lqr1
limit (-10, 10)
PID(s)
outer PID
1
PID(s)
inner PID1
LQR
attitude controller
plant, approximated as double integrators

Linear Quadratic Regulator (LQR) Design



- LQR is a controller that provides optimally controlled feedback gains to stabilize a system
- Applicable when the system is linear and the cost function is quadratic

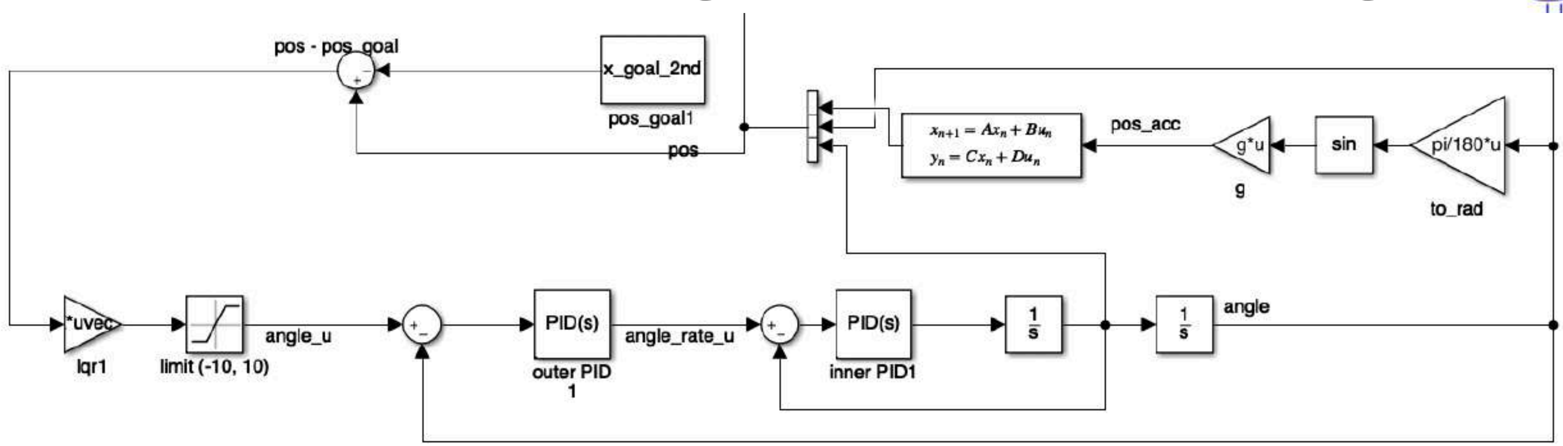
$$J = \sum_{\tau=0}^{\infty} \left(\tilde{\boldsymbol{x}}_\tau^T \mathbf{Q} \tilde{\boldsymbol{x}}_\tau + \boldsymbol{u}_\tau^T \mathbf{R} \boldsymbol{u}_\tau \right) \qquad \boldsymbol{u}_k = \mathbf{K}_{lqr} \tilde{\boldsymbol{x}}_k$$

- The optimal gain is the solution to the algebraic Riccati equation:

$$\mathbf{P} = \mathbf{Q} + \mathbf{A}^T\mathbf{P}\mathbf{A} - \mathbf{A}^T\mathbf{P}\mathbf{B}\left(\mathbf{R} + \mathbf{B}^T\mathbf{P}\mathbf{B}\right)^{-1}\mathbf{B}^T\mathbf{P}\mathbf{A}$$

$$\mathbf{K}_{lqr} = -\left(\mathbf{R} + \mathbf{B}^T\mathbf{P}\mathbf{B}\right)^{-1}\mathbf{B}^T\mathbf{P}\mathbf{A}$$

Linear Quadratic Regulator (LQR) Design



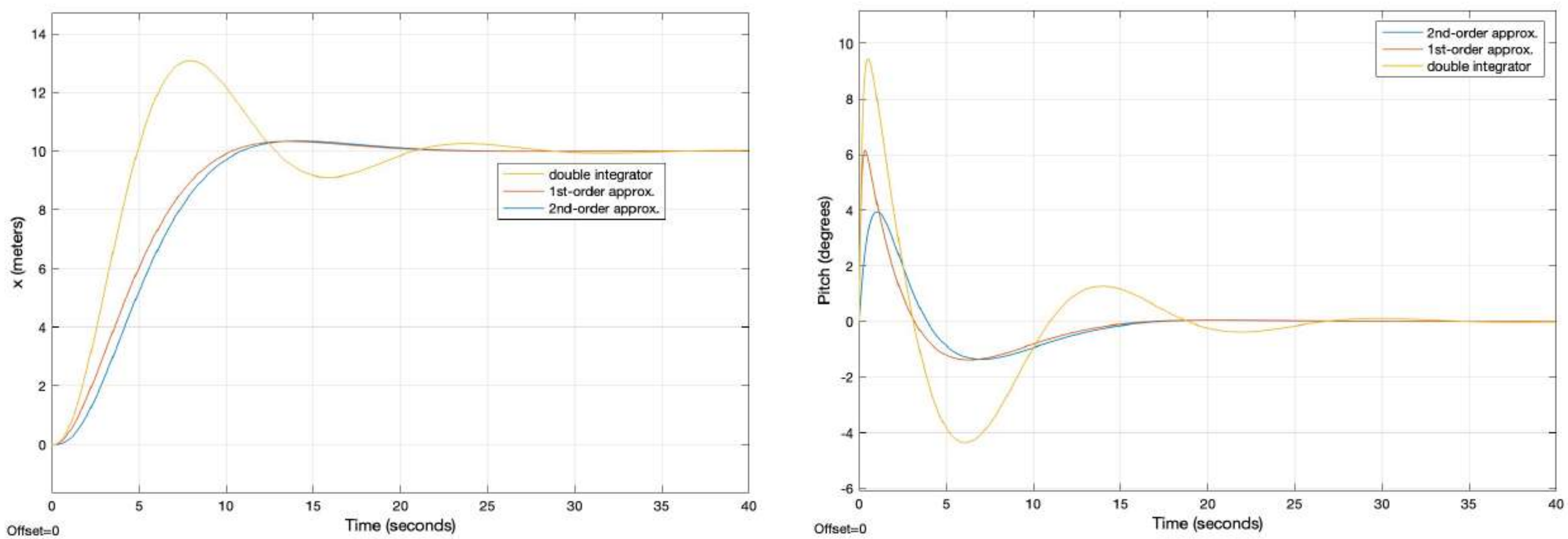
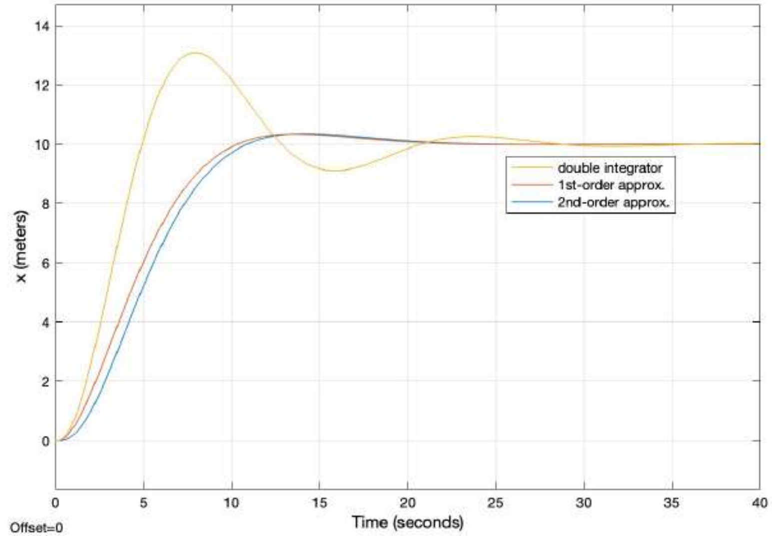
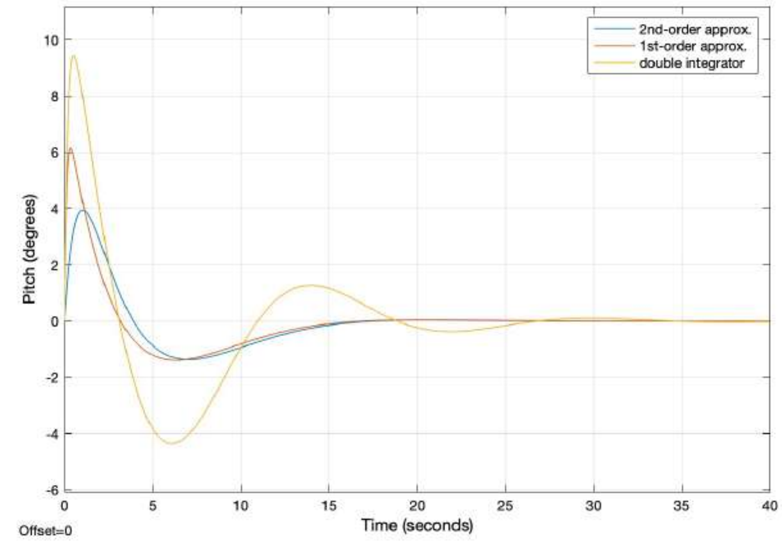


Figure 13: Position and angle of the closed-loop system.

Linear Quadratic Regulator (LQR) Design





- In comparison to double-integrator dynamics:
  - Smaller overshoot
  - Smaller maximum angle
  - Longer risetime

- Proceed with the first-order approximation

Fine Tuning for Critical Damping

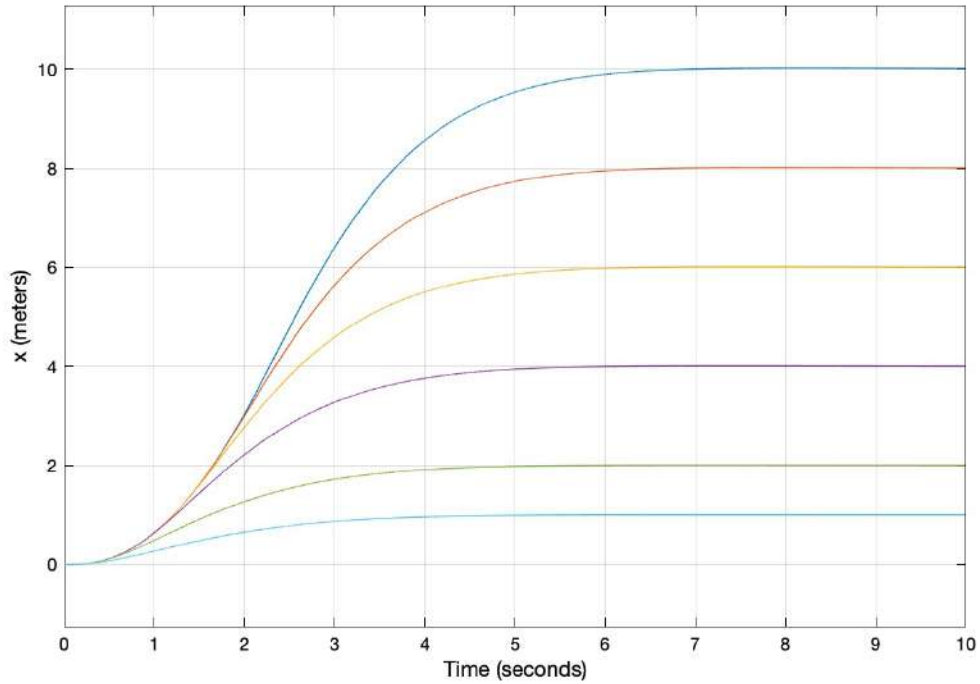


Figure 13: Position of the critically damped closed-loop system.

$$\mathbf{Q} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.01 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.01 \end{bmatrix}$$

$$\mathbf{R} = \begin{bmatrix} 0.01 & 0 \\ 0 & 0.01 \end{bmatrix}$$

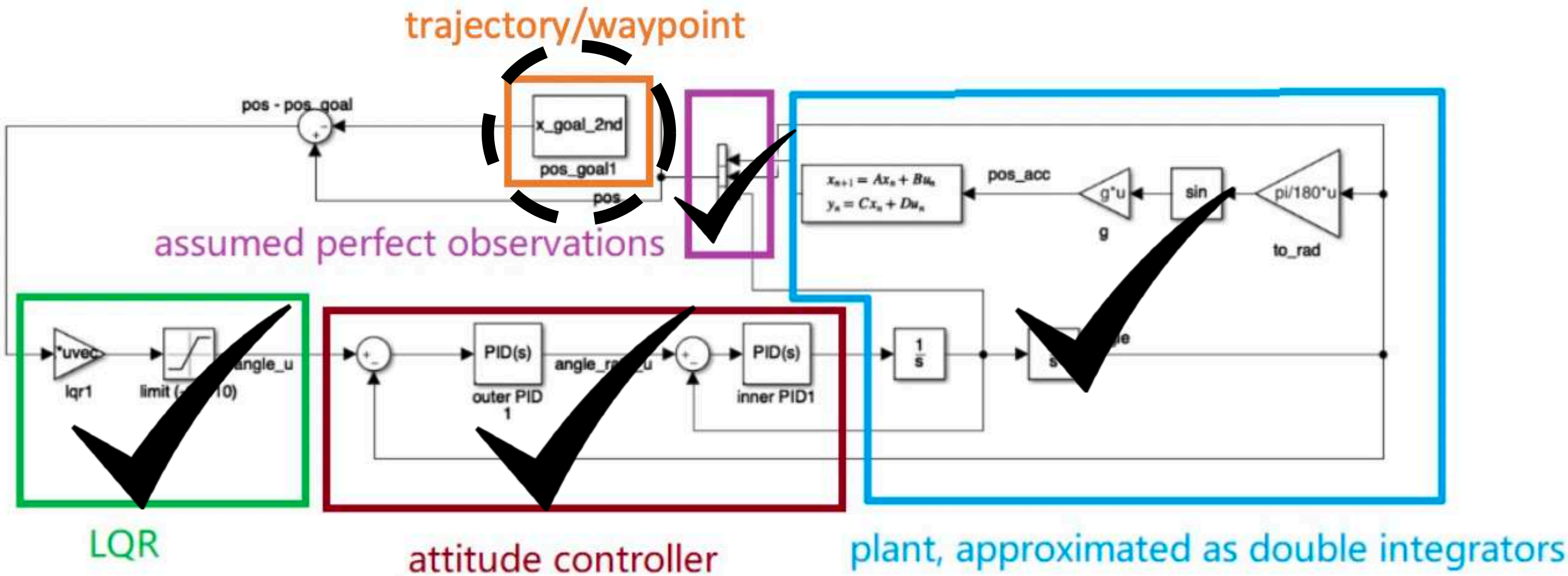
$$\mathbf{K}_{lqr} = \begin{bmatrix} 0 & 10 & 0 & 17.4568 & -0.7527 & 0 \\ -10 & 0 & -17.4568 & 0 & 0 & -0.7527 \end{bmatrix}$$

Yaw and Thrust

- Yaw
  - In simulation, assumed to be 0
  - In experiment, controlled with a simple PD controller and converges to 0 in minimal time
- Thrust
  - Not applicable in simulation, height controlled with a PID controller
  - In experiment, cascade PID for height, the output acceleration is mapped to the desired thrust with a linear mapping

$$thrust = (a_z + 1)\frac{(max\_thrust - min\_thrust)}{2} + min\_thrust$$

  - Future improvement



trajectory/waypoint
pos - pos_goal
x_goal_2nd
pos_goal1
pos
assumed perfect observations
pos_acc
g*u
g
sin
pi/180*u
to_rad
lqr1
angle_u
PID(s)
outer PID
1
PID(s)
inner PID1
LQR
attitude controller
plant, approximated as double integrators

Hunting/Encirclement Criteria

- A platonic solid is a regular, convex polyhedron

Figure 14: Platonic solids in three dimensions.

- Goal is to control the agents so that they eventually form a platonic solid around the target
  - Four agents for this thesis
  - Method to be presented is generalizable to an arbitrary number of agents
- For a tetrahedron centered at (x_t, y_t, z_t) , the Cartesian coordinates of the vertices are:

$$V_1 = (x_t + l, y_t + l, z_t + l)$$

$$V_2 = (x_t + l, y_t - l, z_t - l)$$

$$V_3 = (x_t - l, y_t + l, z_t - l)$$

$$V_4 = (x_t - l, y_t - l, z_t + l)$$

The Hungarian Algorithm

- Assign vertices to the agents in a way that minimizes trajectory crossing
- Hungarian Algorithm can solve the assignment problem in polynomial time. The cost matrix is:

$$C = (-1)\begin{bmatrix} a_1 \cdot v_1 & a_1 \cdot v_2 & a_1 \cdot v_3 & a_1 \cdot v_4 \\ a_2 \cdot v_1 & a_2 \cdot v_2 & a_2 \cdot v_3 & a_2 \cdot v_4 \\ a_3 \cdot v_1 & a_3 \cdot v_2 & a_3 \cdot v_3 & a_3 \cdot v_4 \\ a_4 \cdot v_1 & a_4 \cdot v_2 & a_4 \cdot v_3 & a_4 \cdot v_4 \end{bmatrix} = (-1)\begin{bmatrix} cos(\theta_{11}) & cos(\theta_{12}) & cos(\theta_{13}) & cos(\theta_{14}) \\ cos(\theta_{21}) & cos(\theta_{22}) & cos(\theta_{23}) & cos(\theta_{24}) \\ cos(\theta_{31}) & cos(\theta_{32}) & cos(\theta_{33}) & cos(\theta_{34}) \\ cos(\theta_{41}) & cos(\theta_{42}) & cos(\theta_{43}) & cos(\theta_{44}) \end{bmatrix}$$

- $[C]_{ij}$ is the cost of assigning vertex *j* to agent *i*
- The optimal assignment minimizes the following cost

$$J = \sum_i \sum_j [C]_{ij}[X]_{ij}$$

Design #1: Offline Waypoints Generation + Pure Pursuit

❑ Algorithm

For each (agent, vertex) pair, perform the following steps in the plane containing the vertex vector and the agent vector:

1. Create N equally-spaced circles centered at the target, with the first going through the agent and the last going through the vertex
2. The angular difference from $\mathbf{v}_t^a$ to $\mathbf{v}_t^v$ is divided by N, represented by the dashed lines
3. The *i*th waypoint is the intersection of the *i*th circle with the *i*th dashed line
4. Use a pure pursuit controller for line following

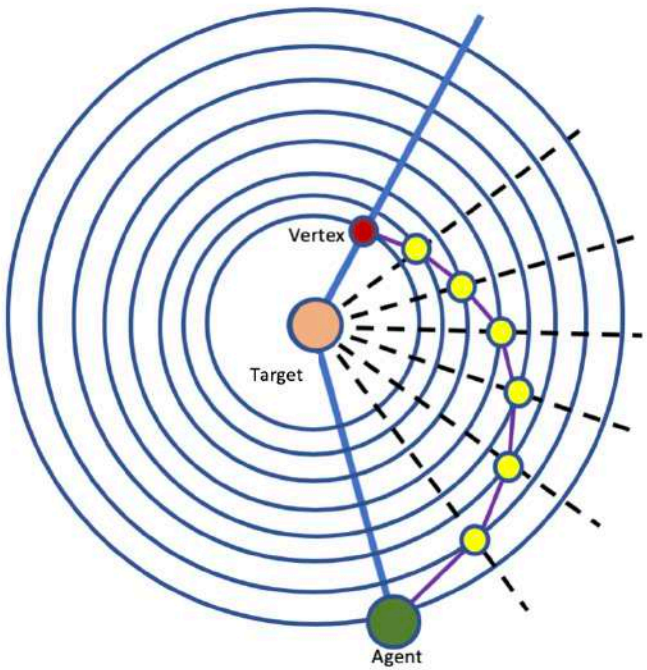


Figure 15: Diagram of Design #1.

Design #1: Offline Waypoints Generation + Pure Pursuit

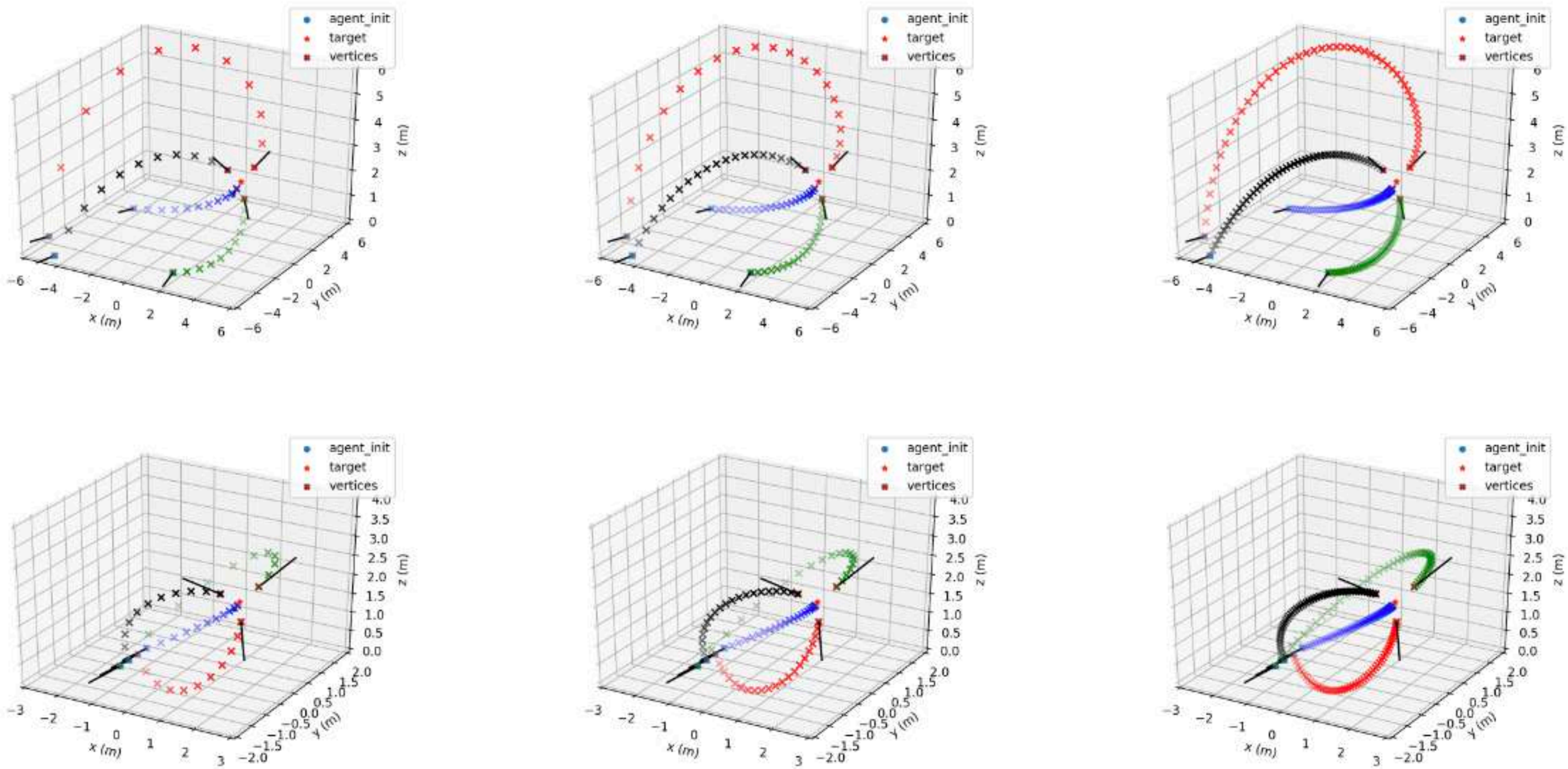


Figure 16: Trajectory generated using 10, 20 and 50 waypoints.

Design #1: Offline Waypoints Generation + Pure Pursuit

- Pure Pursuit line follower
  - Exploits geometric relationships
  - Gently brings the vehicle towards the path
  - Works well when the maneuver isn't aggressive

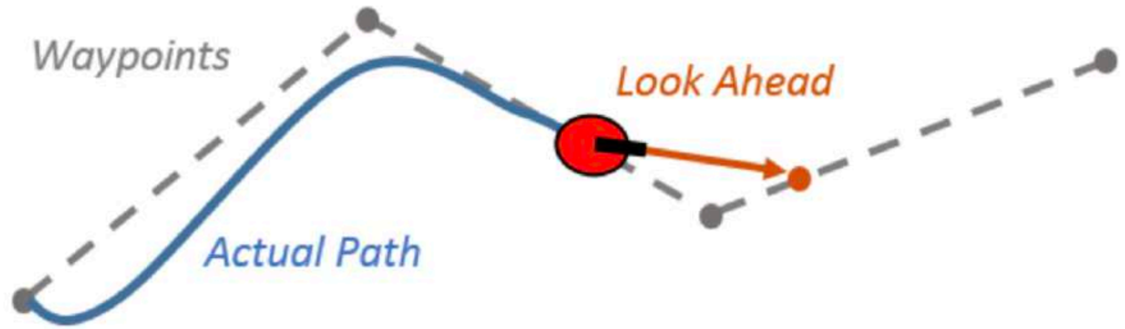


Figure 17: Pure pursuit line follower.

Design #1: Offline Waypoints Generation + Pure Pursuit

- Pure Pursuit line follower

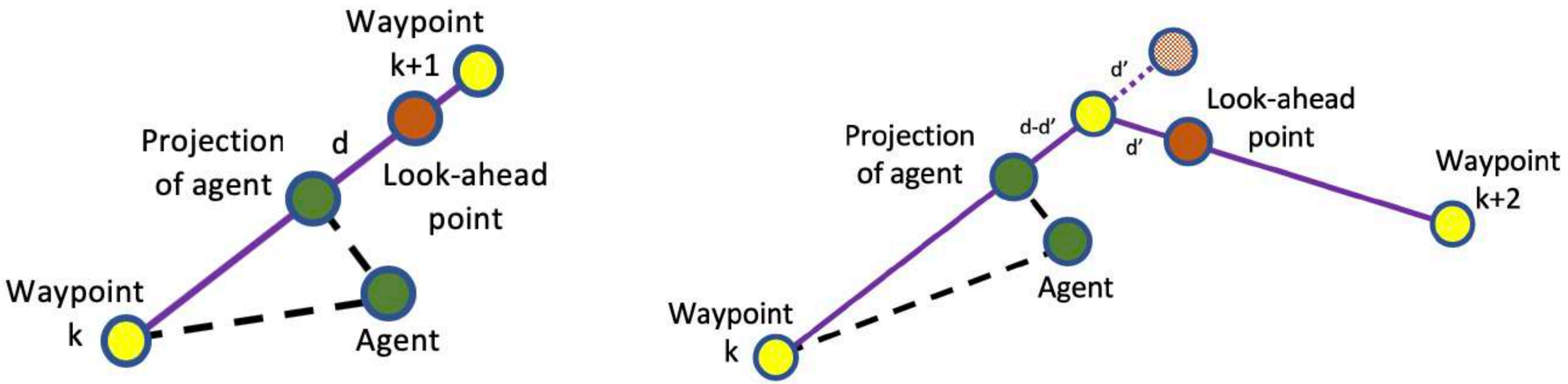


Figure 18: Geometry in the pure pursuit line follower.

Design #1: Offline Waypoints Generation + Pure Pursuit

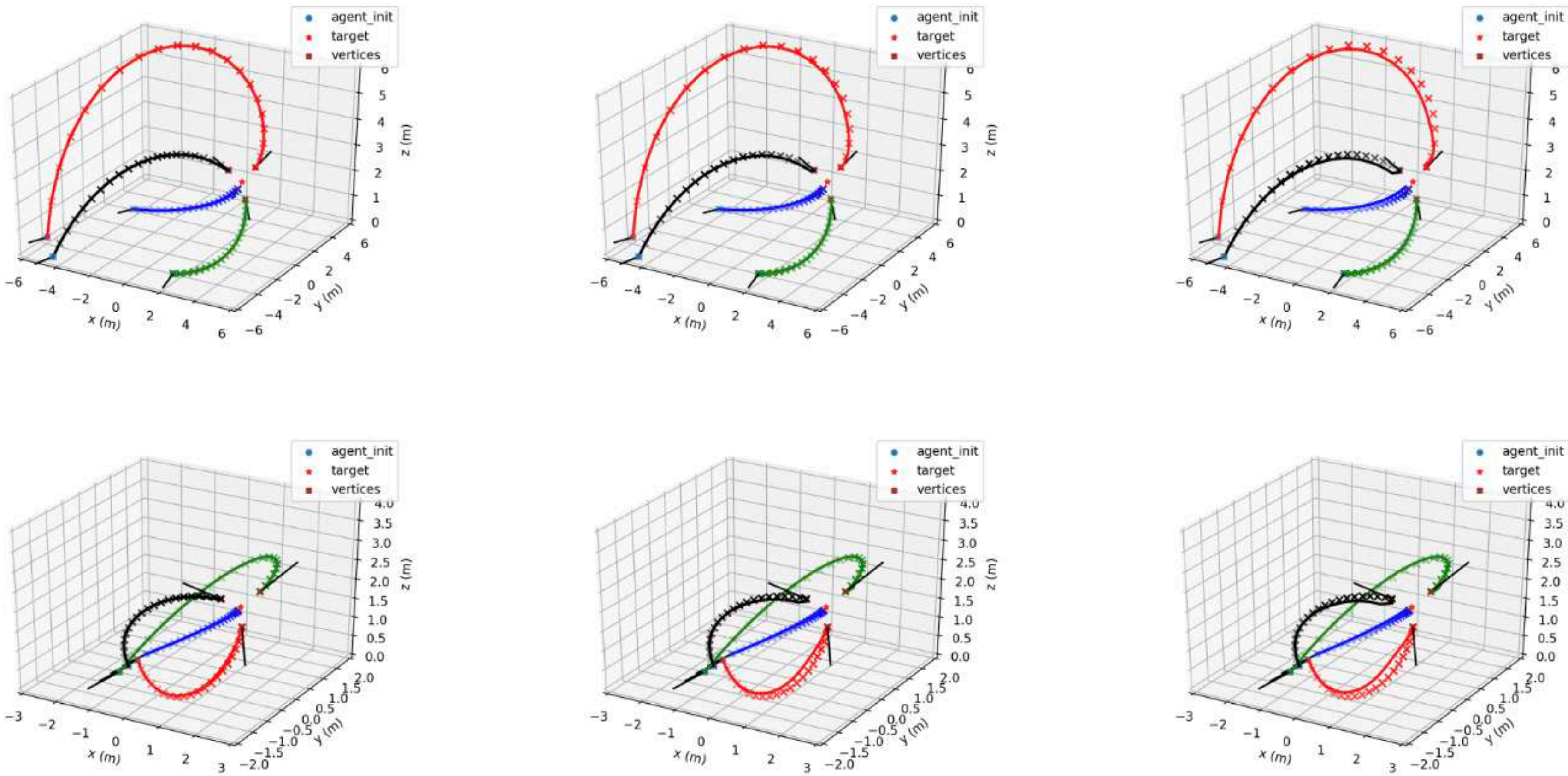


Figure 19: Pure pursuit with 20 waypoints and a look-ahead distance of 5, 15, and 30 cm.

Discussion

- Requires an ideal environment to work well
- The offline generation of the next waypoint assumes perfect tracking of the current waypoint.
  - Downwash when trajectories cross
  - Reactive collision avoidance
- Design #1 results in perfect encirclement trajectories and can serve as a baseline

- The next waypoint generated using the current position of the agent will be more optimal
- This leads us to Design #2

Design #2: Online Waypoints Generation

- Now generate the waypoint on the fly
- Algorithm

For each (agent, vertex) pair, perform the following steps in the plane containing the vertex vector and the agent vector:

1. Hungarian Algorithm for assignment
2. Recalculate the angular and radial difference denoted by θ and D, respectively
3. Then, the look-ahead point is obtained by rotating the agent vector by $d\theta = \alpha_1 \theta$ and shrink its magnitude by $dD = \alpha_2 D$
4. Rinse and repeat

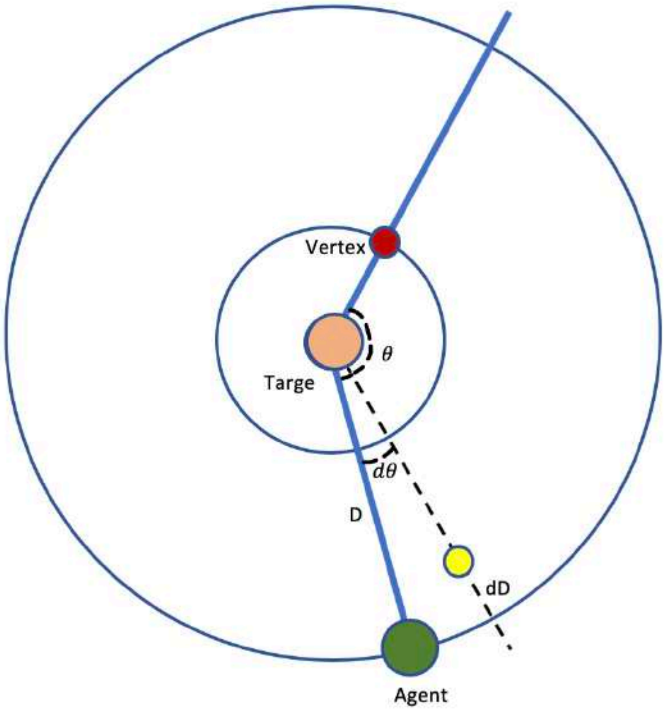


Figure 20: Diagram of Design #2.

Design #2: Online Waypoints Generation

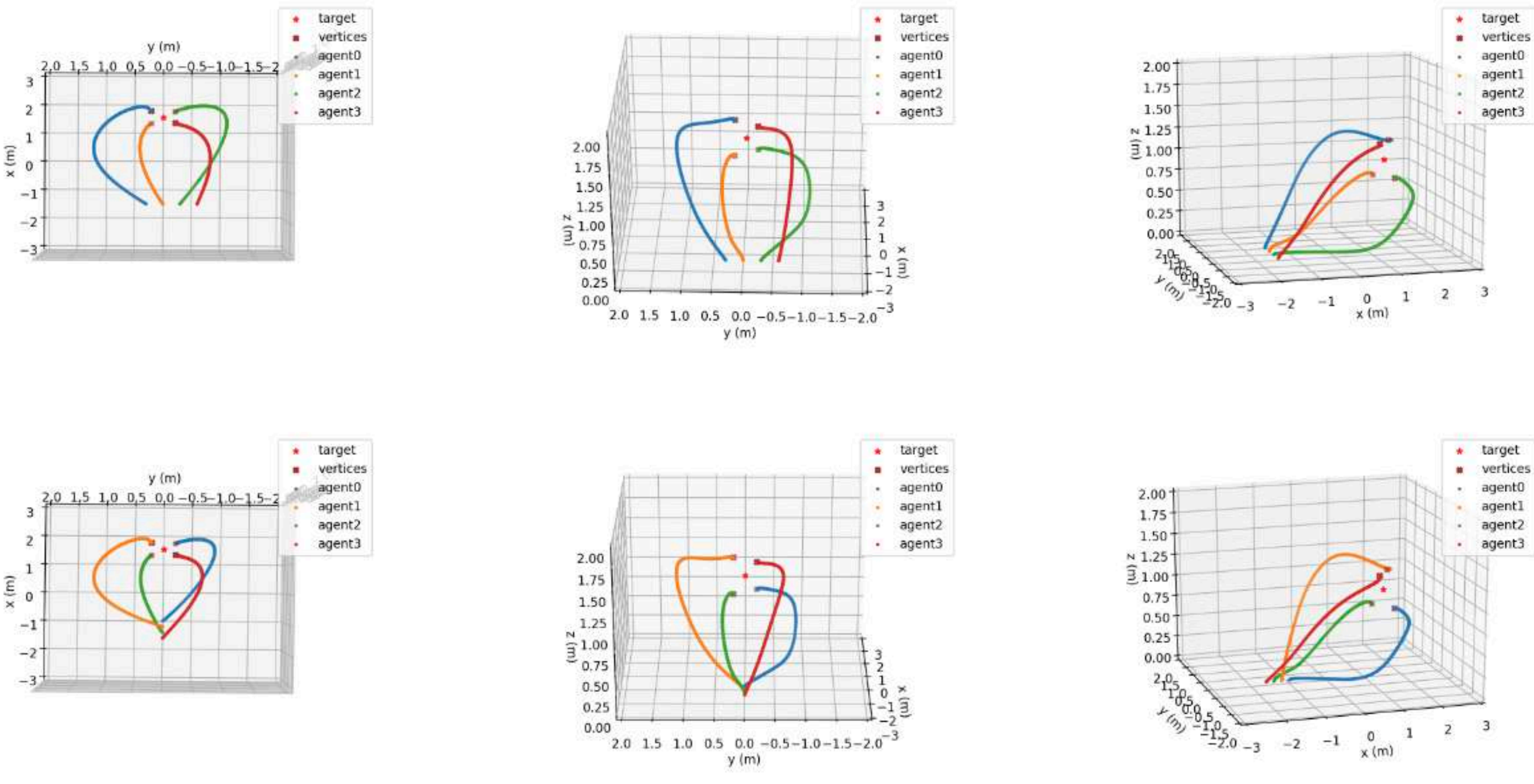


Figure 21: Simulation results of Design #2.

Design #2: Online Waypoints Generation

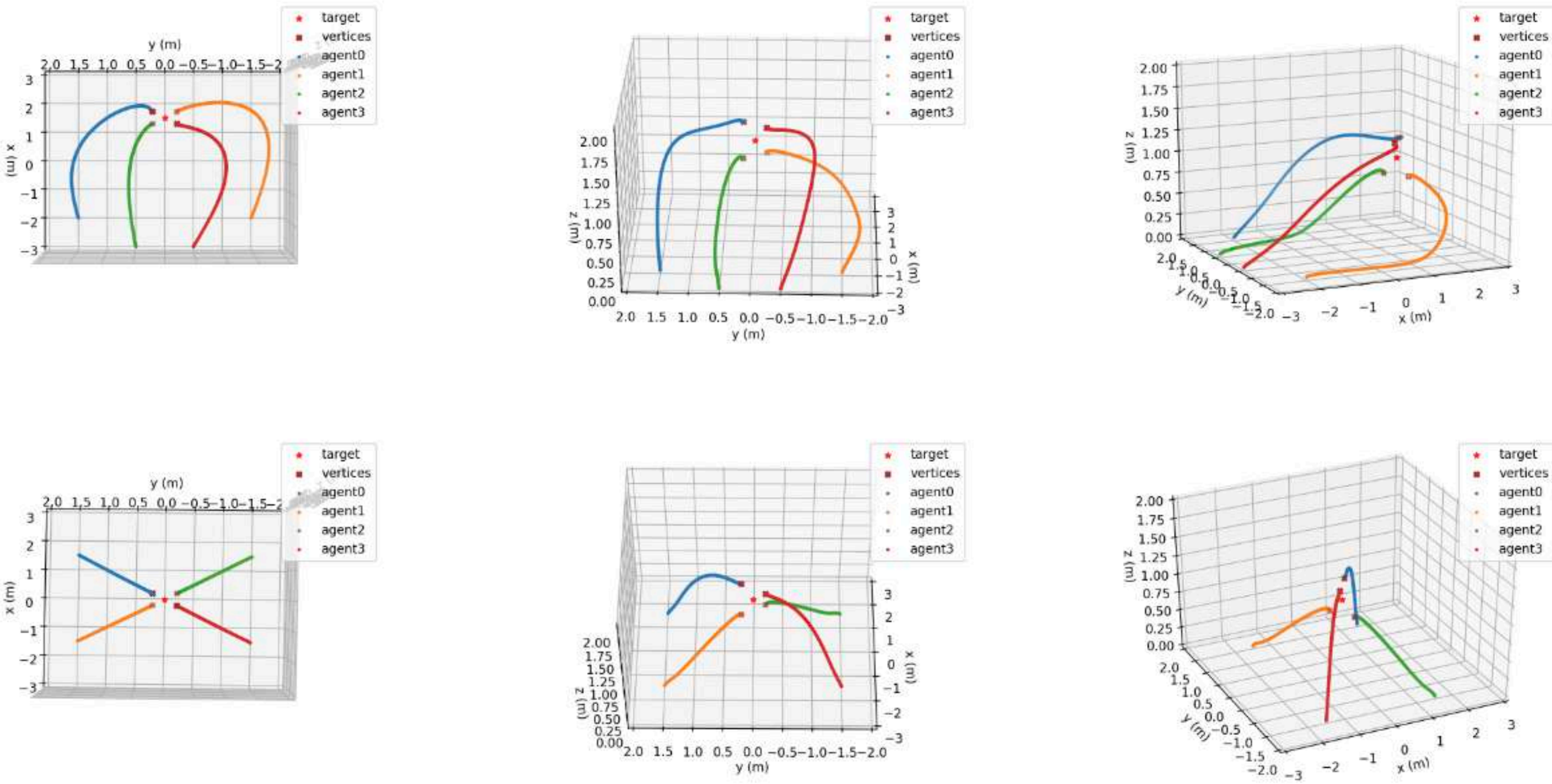
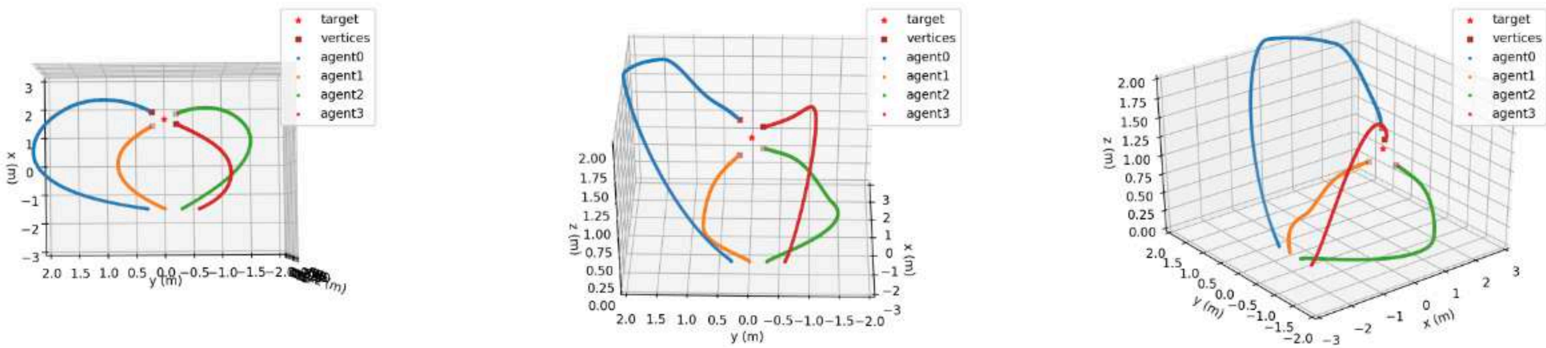


Figure 21: Simulation results of Design #2.

Design #2: Online Waypoints Generation

- Effect of decreasing the alphas



- Effect of increasing the alphas

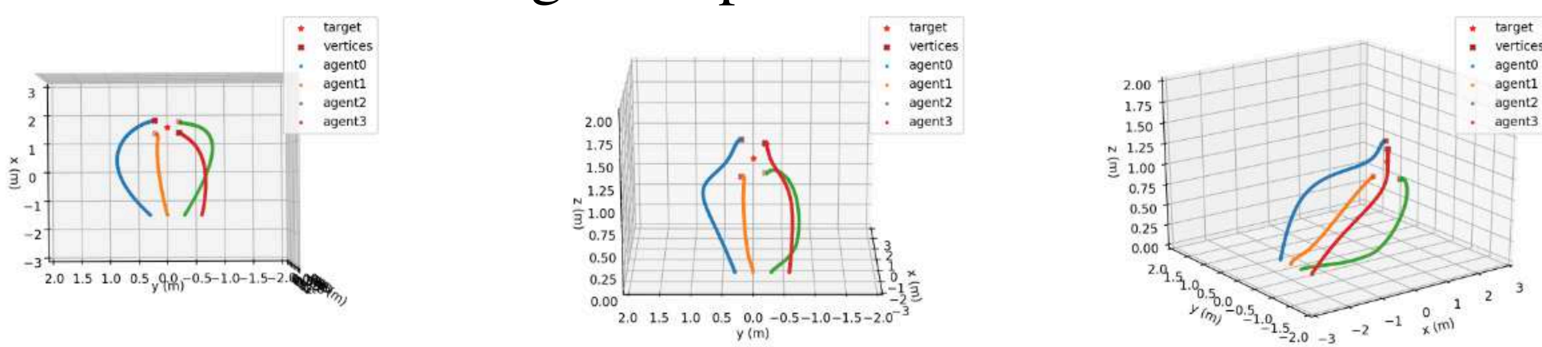


Figure 21: Simulation results of Design #2.

Design #2: Online Waypoints Generation

- Observations
  - Different from Design #1
  - Trajectory for each agent no longer lie entirely in one plane
  - Convergence is faster in the z direction

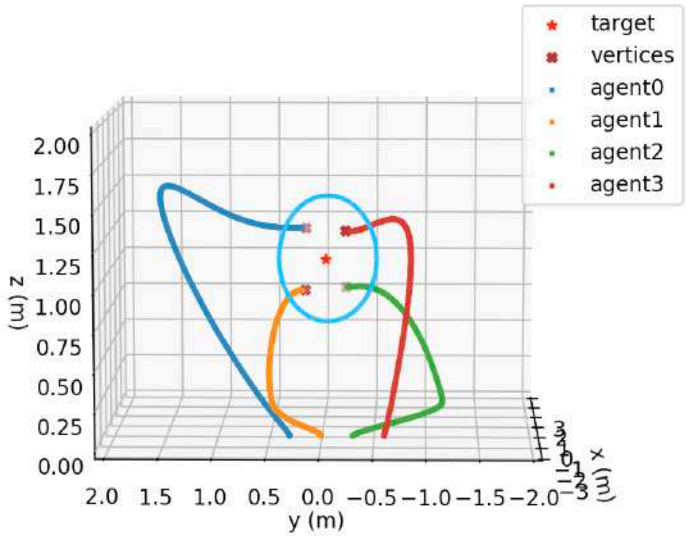
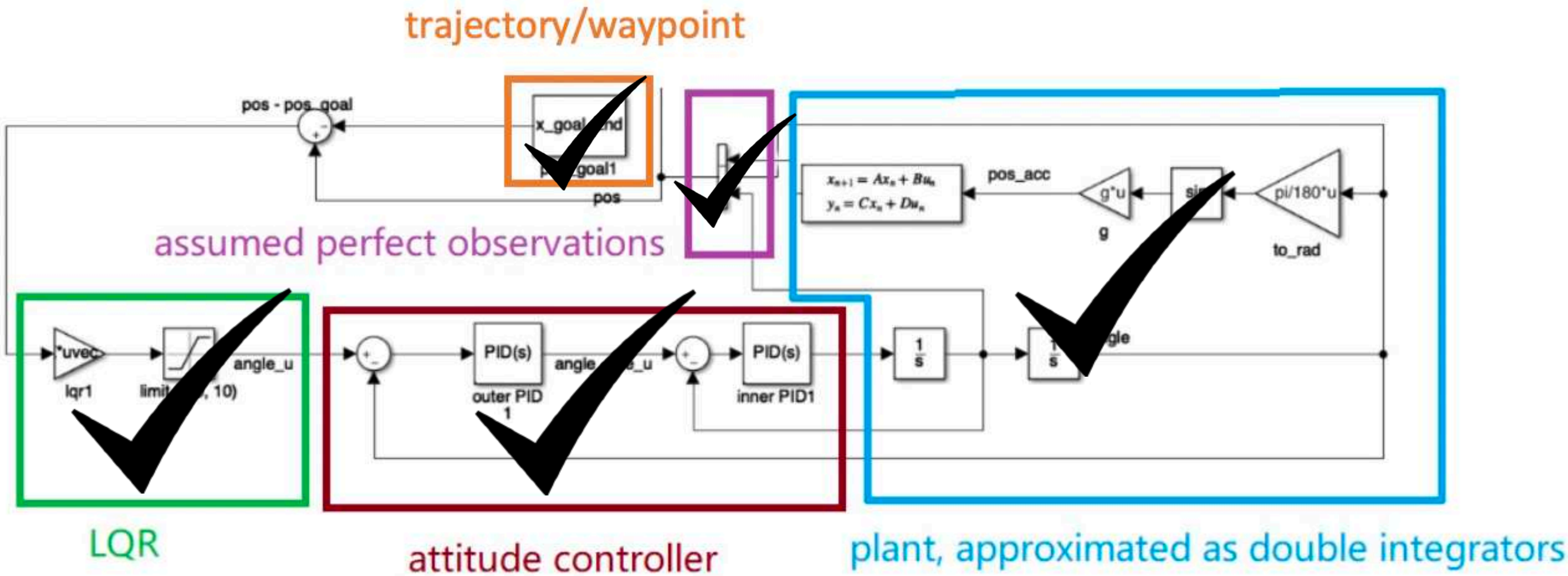


Figure 21: Simulation results of Design #2.

Design #2: Online Waypoints Generation



Experiment Setup

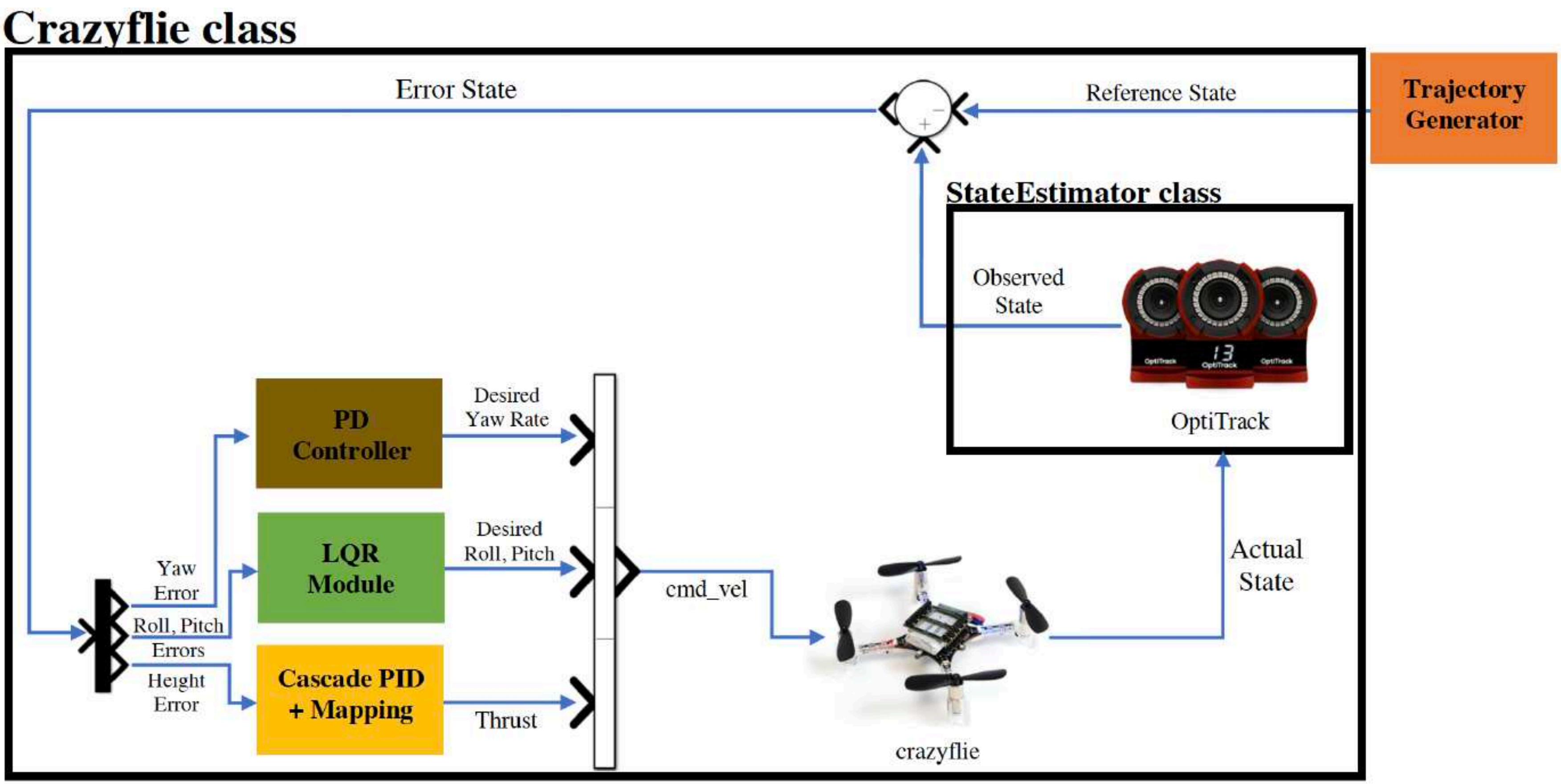


Figure 22: Experiment Setup.

Simplified ROS Architecture

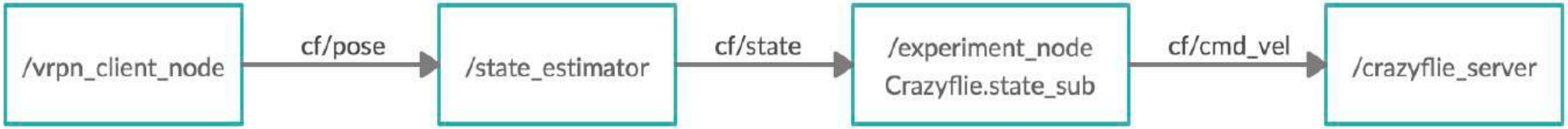


Figure 23: Simplified ROS architecture.

Single-Agent Test

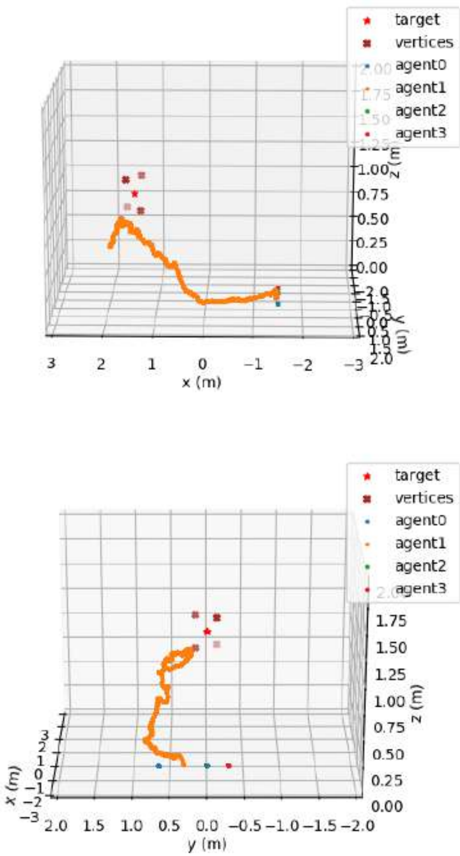


Figure 24: Trajectory using position reference.

$$\bar{x} = \begin{bmatrix} x_d \\ y_d \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Full-state reference using **Spline Interpolation**

$$\bar{x} = \begin{bmatrix} x_d \\ y_d \\ \dot{x}_d \\ \dot{y}_d \\ \phi_d \\ \theta_d \end{bmatrix}$$

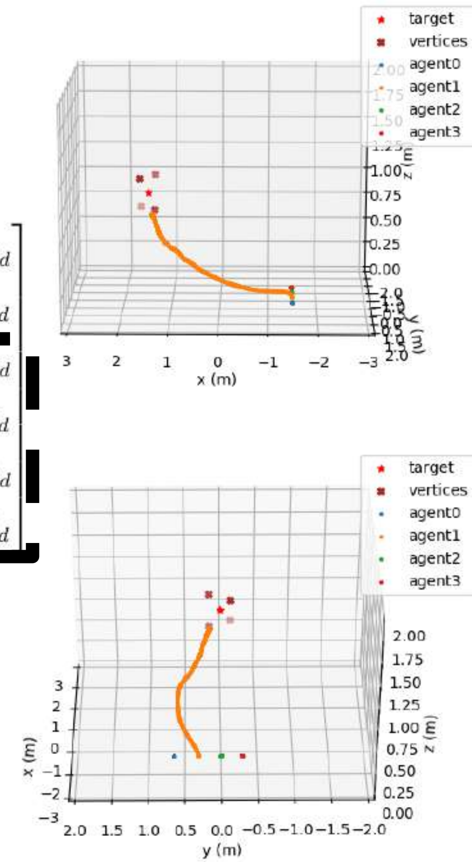


Figure 25: Trajectory using full state reference.

Spline Interpolation

- A spline is a special function defined piecewise by polynomials
  - High accuracy
  - Low computational effort
  - Differentiability
- In our case, the next 10 waypoints are calculated for interpolation

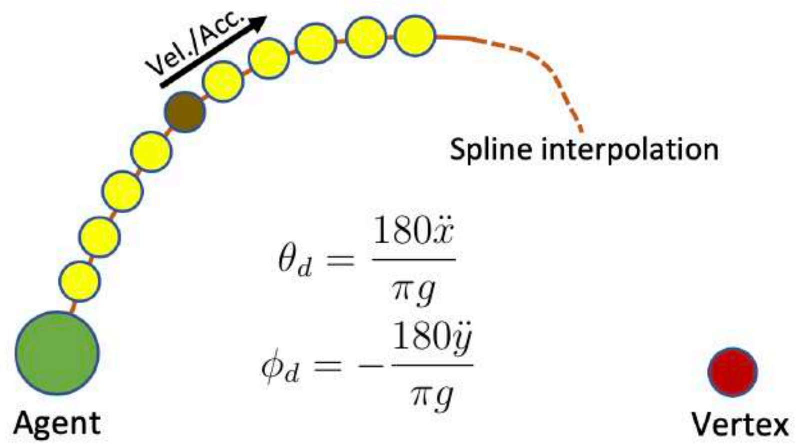


Figure 26: Spline interpolation for velocity and acceleration.

- The position, velocity, and acceleration at 5*dt = 0.5 s are used for the reference state
- Look-ahead point

Collison Avoidance

- Based on *Flocking for Multi-Agent Dynamic Systems: Algorithm and Theory* by Olfati-Saber

$$\mathbf{u}_i = c_1 \sum_{j \in N_i} \phi_\beta \left(\|\mathbf{q}_j - \mathbf{q}_i\|_\sigma \right) \mathbf{n}_{i,j} + c_2 \sum_{j \in N_i} b_{i,j} \left(\mathbf{p}_j - \mathbf{p}_i \right)$$

- Plot of $\phi_\beta(z) = \rho_h \left(\frac{z}{d_\beta} \right) \left(\sigma_1 \left(z - d_\beta \right) - 1 \right)$

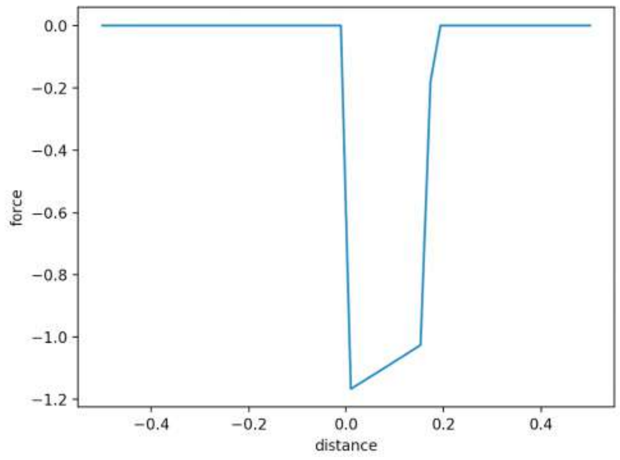


Figure 27: Plot of $\phi_\beta(z)$.

Collison Avoidance

Away from agent j Same direction as agent j

$$\mathbf{u}_i = c_1 \sum_{j \in N_i} \boxed{\phi_\beta \left(\|\mathbf{q}_j - \mathbf{q}_i\|_\sigma \right) \mathbf{n}_{i,j}} + c_2 \sum_{j \in N_i} \boxed{b_{i,j} \left(\mathbf{p}_j - \mathbf{p}_i \right)}$$

- Plot of $b_{i,k}(q) = \rho_h \left(\frac{\|q_k - q_i\|_\sigma}{d_\beta} \right)$, $\rho_h(z) = \begin{cases} 1 & \textbf{if } z \in [0, h) \\ \frac{1}{2}\left[1 + \cos\left(\pi \frac{(z-h)}{(1-h)}\right)\right] & \textbf{if } z \in [h, 1] \\ 0 & \textbf{otherwise} \end{cases}$

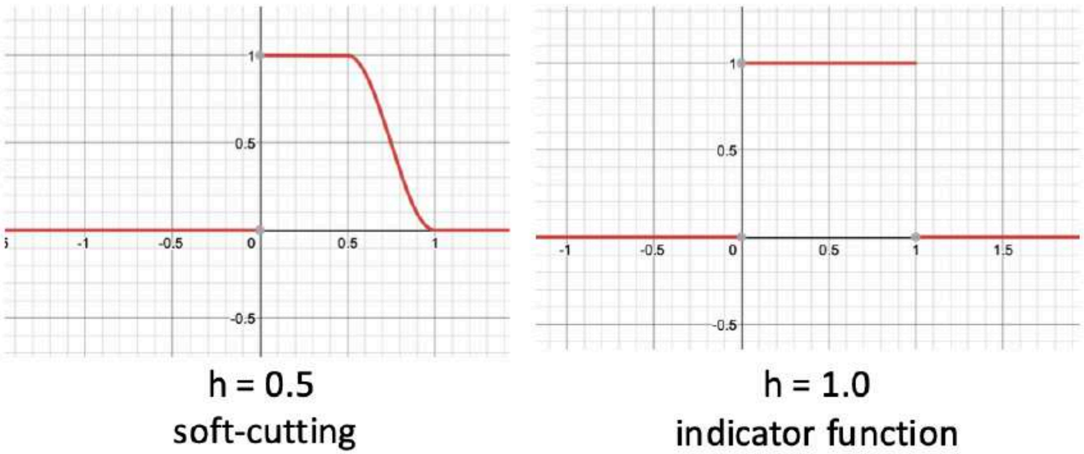


Figure 28: Plot of $\rho_h(z)$.

Collison Avoidance

- Simulation results

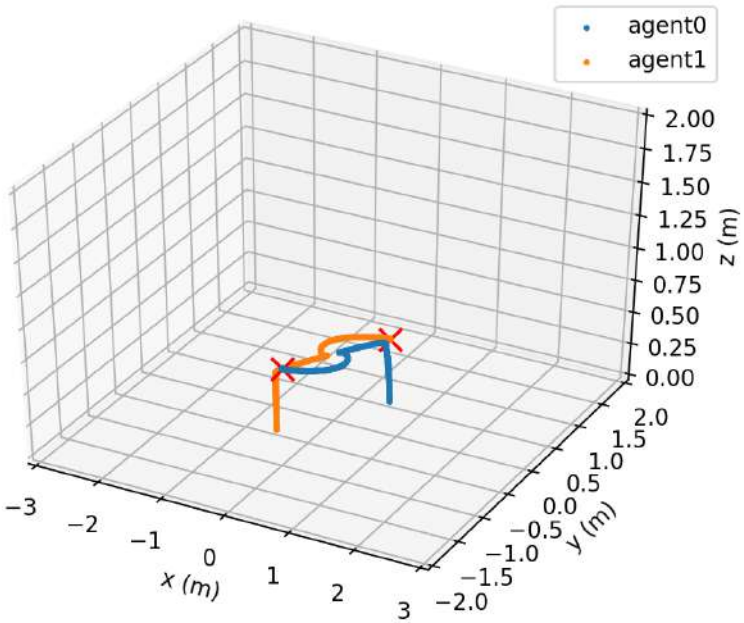


Figure 29: Collision test looking from the above.

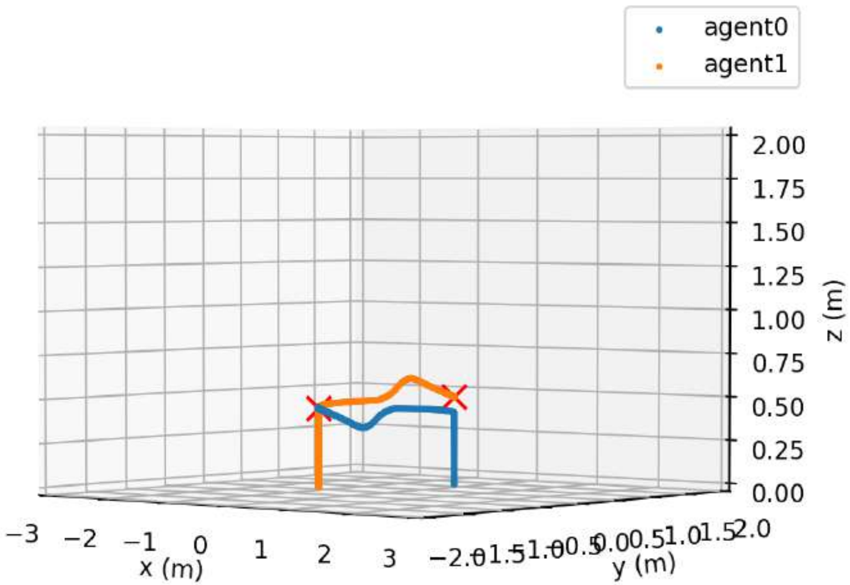


Figure 30: Collision test looking from the side.

Collison Avoidance

- Experiment results

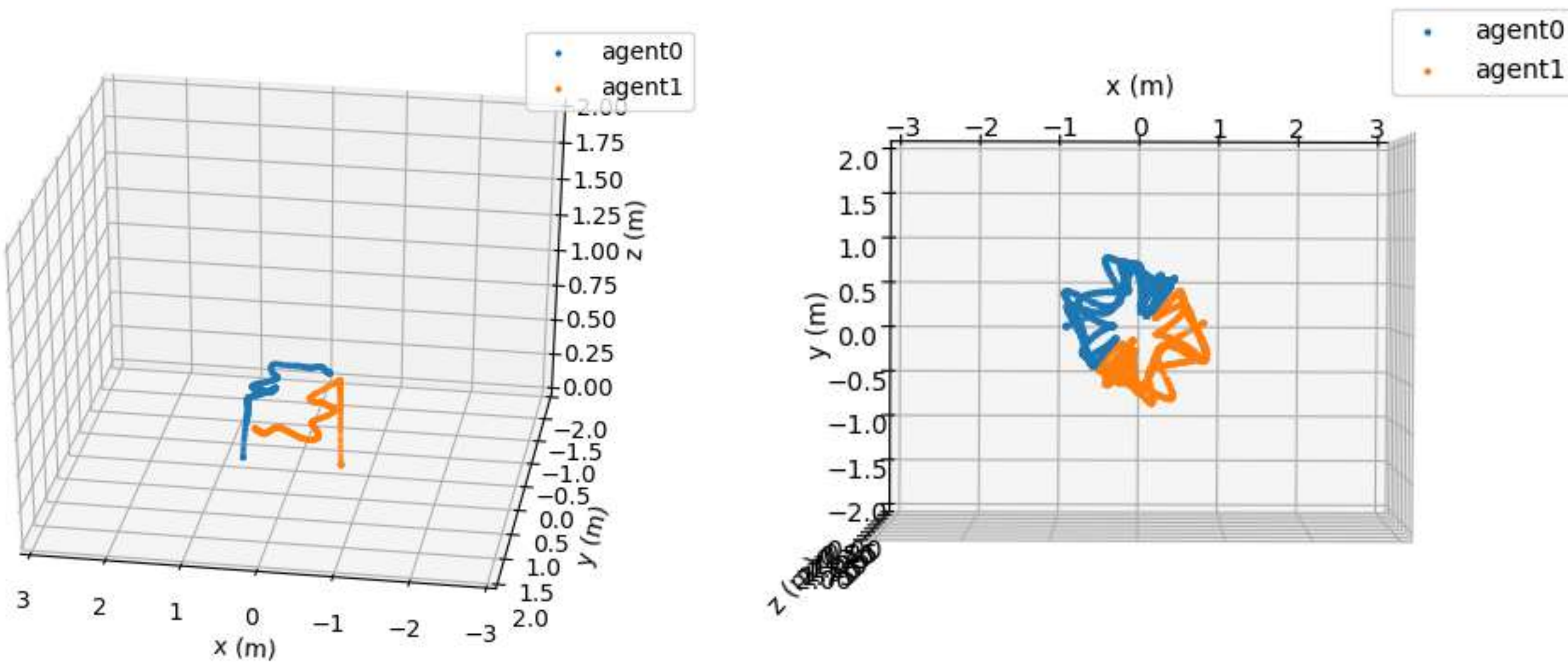
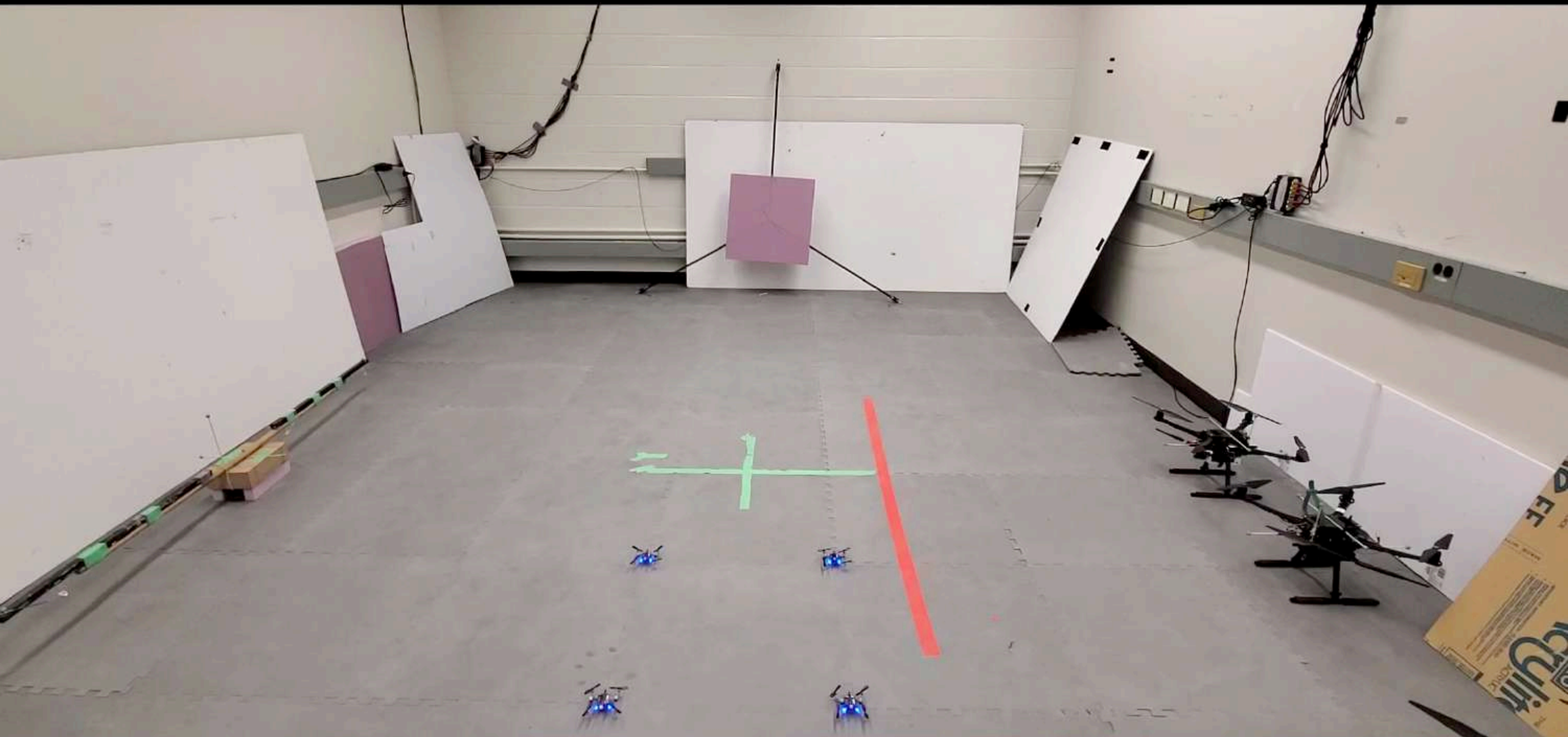
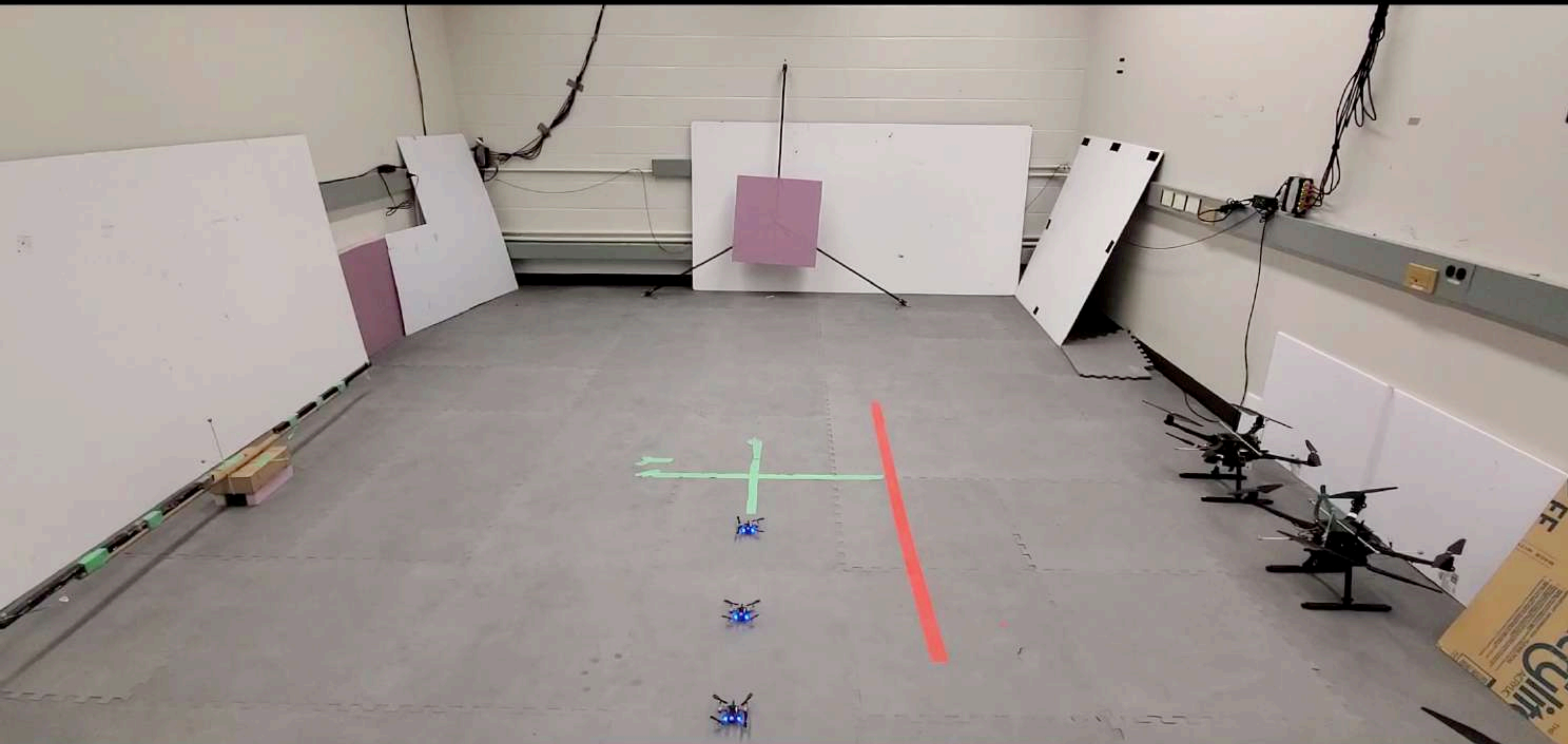


Figure 31: Collision avoidance experiments.

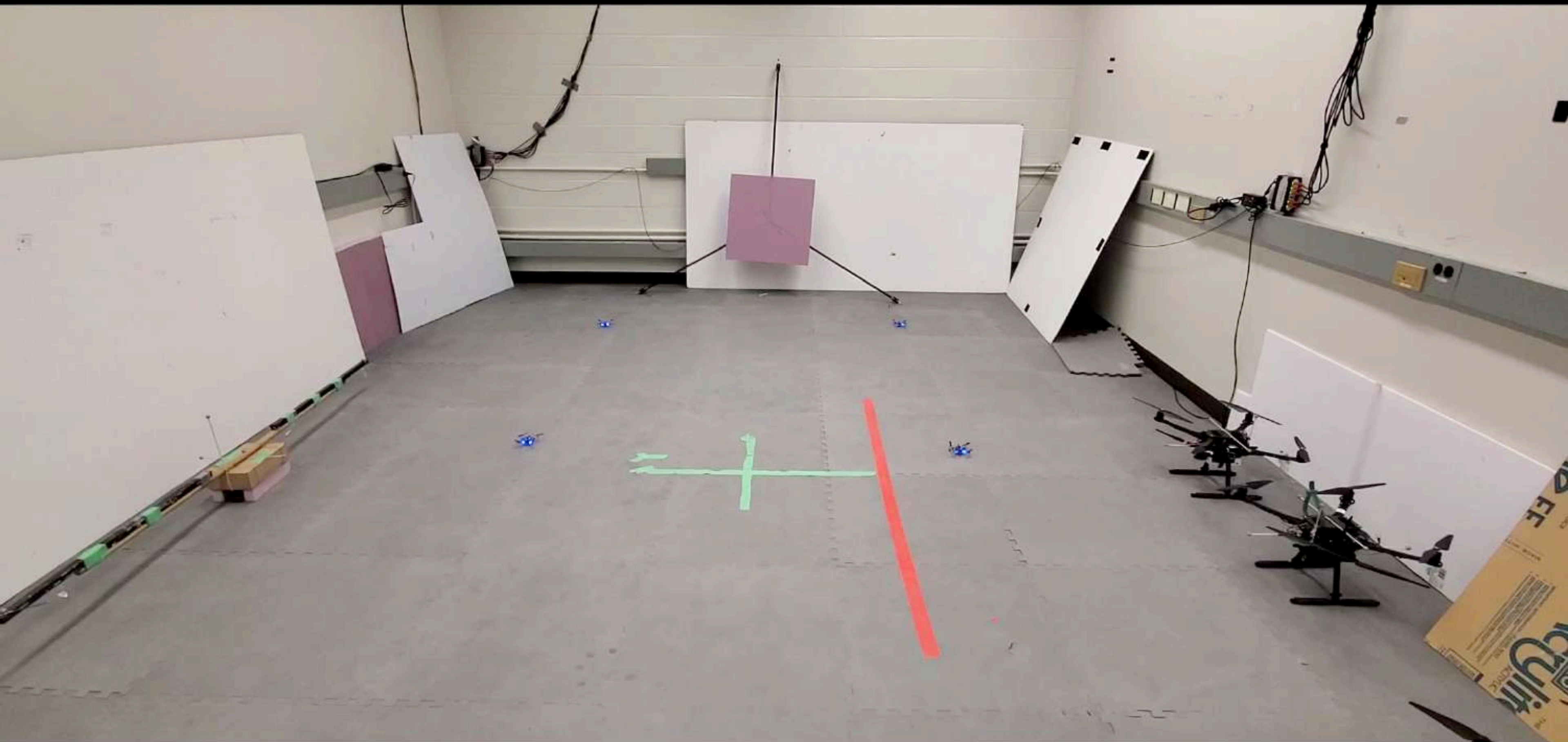
Multi-agent Hunting



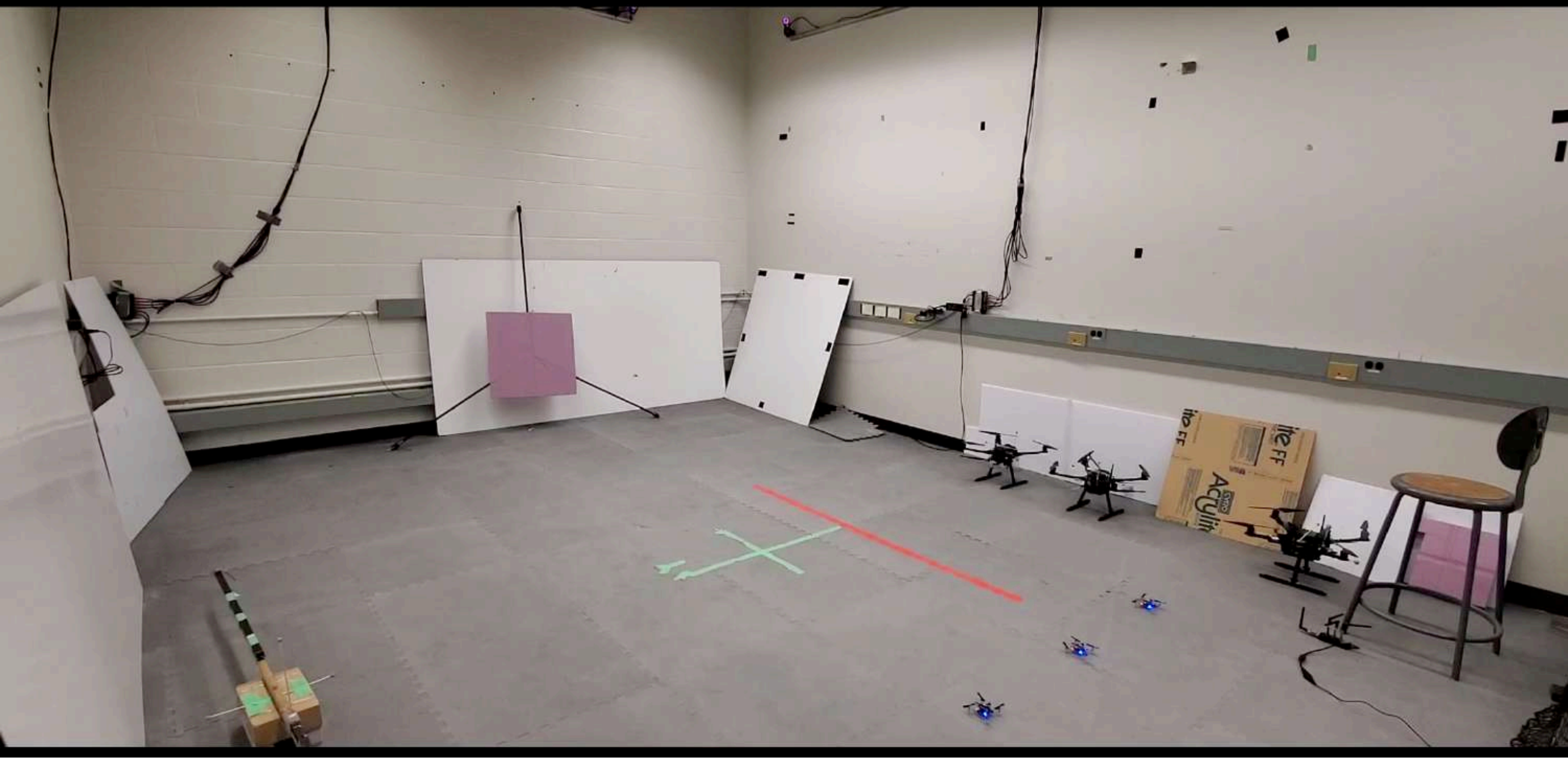
Multi-agent Hunting



Multi-agent Hunting



Multi-agent Hunting



Multi-agent Cooperative Hunting

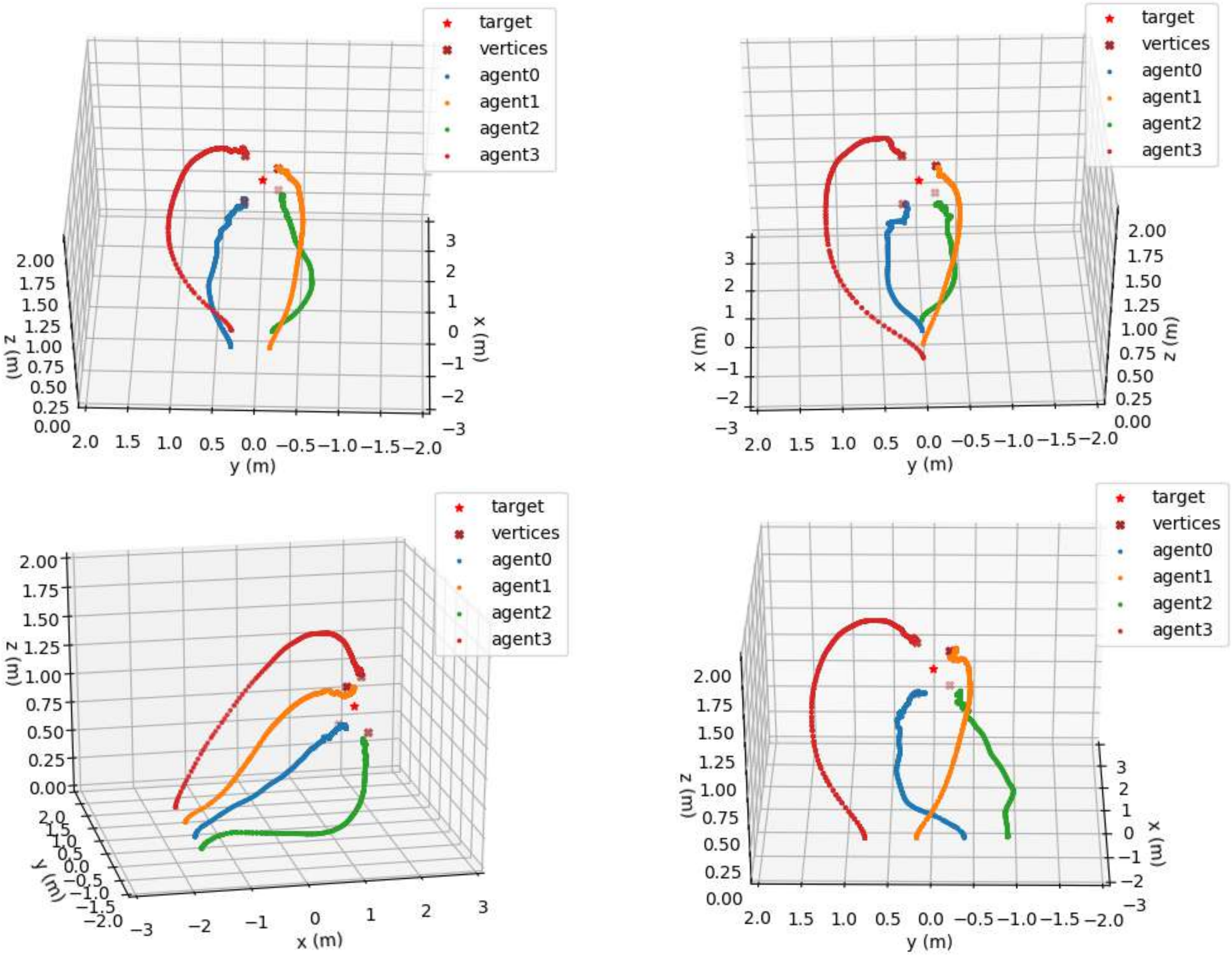


Figure 32: Multi-agent cooperative hunting experiments.

Cooperative Hunting with Moving Target

- ❑ The algorithm can be extended to handle moving target
- ❑ Use the Crazyflie dynamics for the target as well
- ❑ Vertices are no longer fixed
- ❑ Programmable ring deck
  - o Red → target

Figure 34: Crazyflie quadcopter with LED ring deck.

Moving Target – Simulation

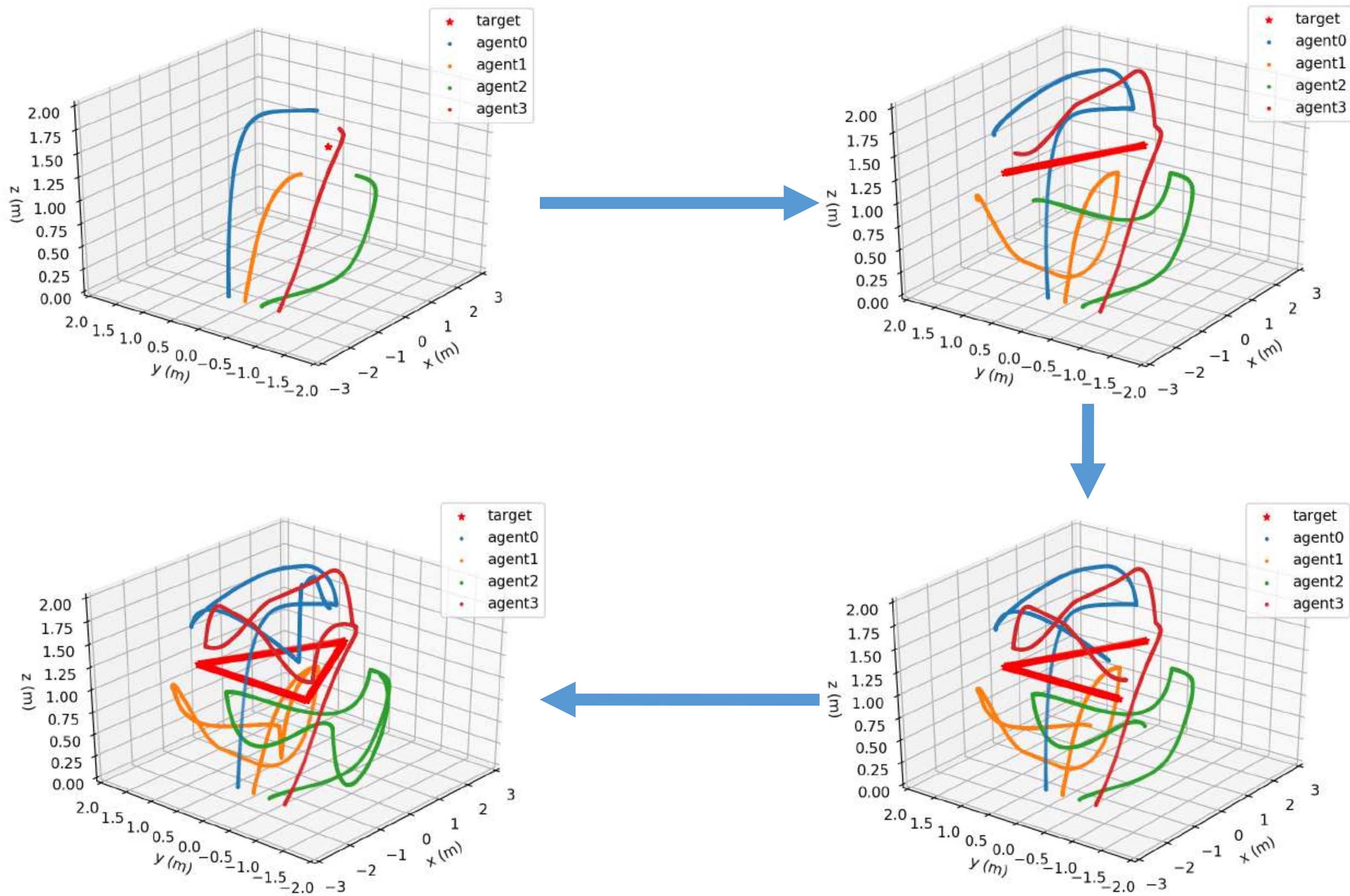
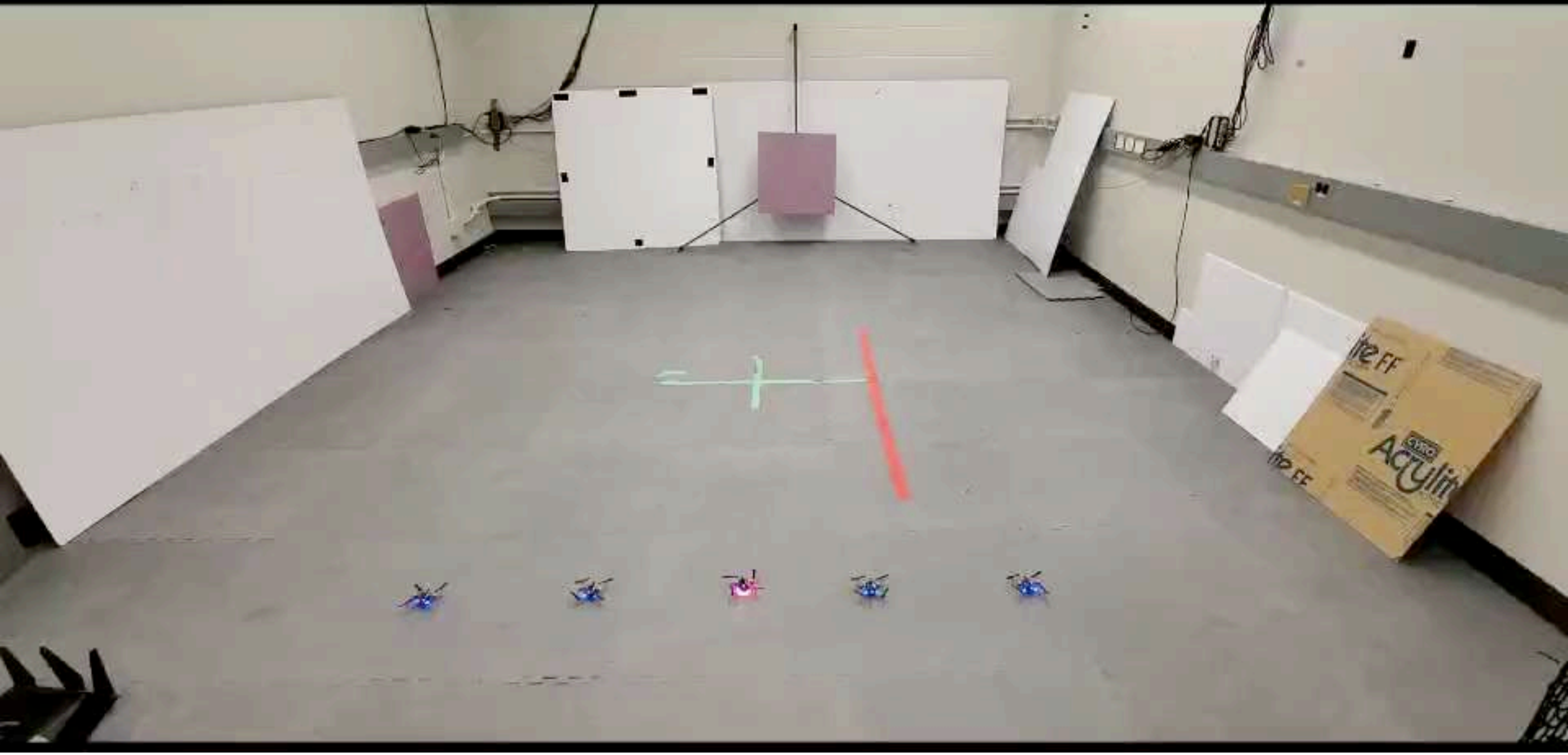


Figure 33: Cooperative hunting with moving target.

Moving Target – Experiment

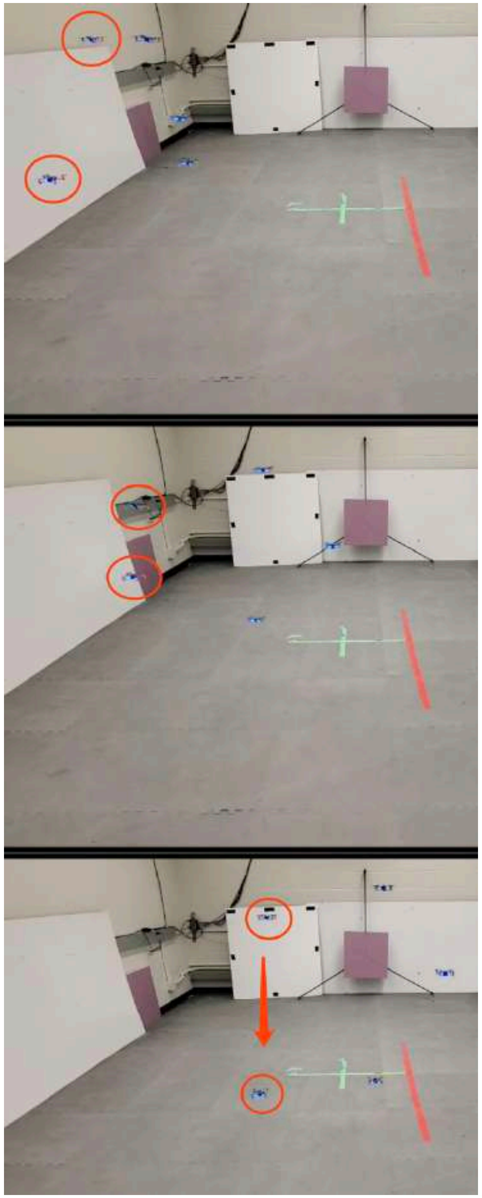


Failure Analysis

- The hunter agents were able to capture the target at the first and second stop

- At one point, two agents decides to swap places due to a change in the assignment

- Trajectories cross → downwash → more assignment change

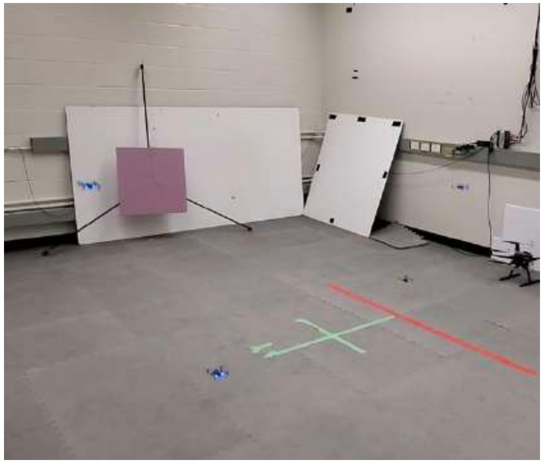


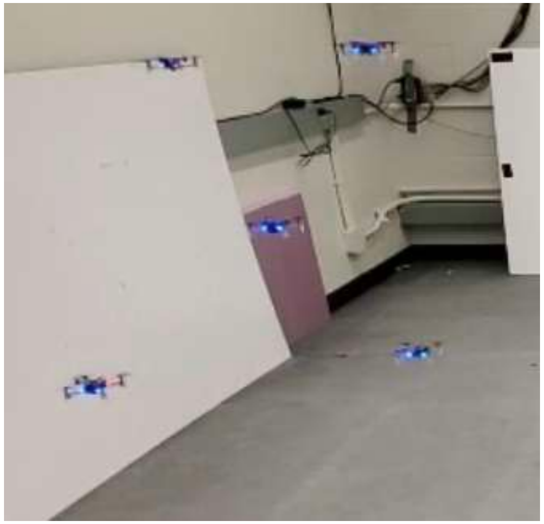
Future Improvement

- Thrust not modeled
  - Double-integrator dynamics in simulation
  - Acceleration to thrust mapping in experiment
- Can get stuck in local minimums when using a reactive collision avoidance controller
  - Add a planner module to plan around obstacles
- Encirclement is not simultaneous
- Trajectories cross
  - Downwash
  - Assignment change
- Add obstacles in the environment

Conclusion

- 3D cooperative hunting algorithm for the undergraduate thesis
- Preliminary experimental results
- Thanks to the Flight System and Control Lab!
  - Thanks to Prof. Liu for the summer research + thesis opportunity
  - Thanks to Jacky for the guidance, hope everything goes well after graduation





Contact: **Che (Charles) Liu**,. Tel: 416-559-9916
Email: charlesliuoft@gmail.com