Using Model Predictive Control for Quadrotor Interception

Benjamin Ung
Carnegie Mellon University
Pittsburgh, Pennsylvania, USA
benjaminung@cmu.edu

***Abstract*—This paper presents a method to use a model predictive control on a quadrotor to protect an area from another quadrotor by intercepting it. The defending quadrotor could be equipped with mechanisms to disable the offending quadrotor, but this paper will not cover that detail. The model predictive control was tested in a computer simulation using the programming language, Julia. Several different trajectories were given to the offending quadrotor. The defending quadrotor only uses the offending quadrotor's current state to determine the interception point, and its model predictive control calculates the optimal trajectory to the interception point.**

***Index Terms*—model predictive control, optimal control, quadrotor, unmanned aerial vehicle**

I. Introduction

Quality Quadrotors are available for anybody to purchase today, and can be used for nefarious purposes. This creates a security concern, especially for law enforcement departments and militaries. There are current solutions to disable or capture threatening quadrotors such as capturing it with a net gun [1], or spraying material to cause the rotors to malfunction [2]. One of these methods could be mounted on a commercially available quadrotor that will intercept the path of the threatening quadrotor. A controller will need to be designed for determining the interception trajectory.

II. Objective

This paper will discuss the implementation of a control system to determine this interception trajectory. The control system and the simulation is written in Julia programming language. Visualization of the simulation is done with the MeshCat package in Julia. In the visualization, the threatening quadrotor is displayed in red and will be referred to as the red quadrotor. The defending quadrotor is blue and will be referred to as the blue quadrotor.

First a trajectories for the red quadrotor are calculated. The simulation advances at discrete time steps, and moves the red quadrotor along its previously calculated trajectory. At each time step, a new interception point is calculated for the blue quadrotor based on the red's current state. Then the blue quadrotor's model predictive control (MPC) will determine the optimal trajectory to this interception point.

Each simulation run lasts 20 seconds with 1001 time steps, each lasting 0.02 seconds. A Cartesian coordinate system was used with the x direction as north/forward, the y direction as west/left, and the z direction as up. The blue quadrotor starts at a position 0, -5, 14 meters, and the red at 0, 100, 100. The blue quadrotor is defending position 0, 0, 0 which is represented as a green box in the MeshCat visualizer.

The source code for this implementation is found in this GitHub repository. This code was based off of lecture notes and code in [3] and [4].

III. Quadrotor Dynamics

Both the red and blue quadrotor have the same dynamics and resemble commercially available quadrotors. The states, $\boldsymbol{x}$, has 13 elements–three for position, four for the quaternion representation of its rotation, three for the velocity in body coordinates, and three for the angular velocity. The controls, u, has 4 elements–the thrust for each rotor.

$$\boldsymbol{x} = \begin{bmatrix} \boldsymbol{p} \\ \boldsymbol{q} \\ {}^B\boldsymbol{v} \\ {}^B\boldsymbol{\omega} \end{bmatrix} = \begin{bmatrix} p_x \\ p_y \\ p_z \\ q_w \\ q_x \\ q_y \\ q_z \\ {}^Bv_x \\ {}^Bv_y \\ {}^Bv_z \\ {}^B\omega_x \\ {}^B\omega_y \\ {}^B\omega_z \end{bmatrix}, \quad \boldsymbol{u} = \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{bmatrix} \tag{1}$$

To calculate the derivative with respect to time, the quaternion needs to be converted to a rotation matrix, $\boldsymbol{Q}$, to rotate the velocity from body to world coordinates. This is described in [4].

$$\boldsymbol{T} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & -1 \end{bmatrix}, \quad \boldsymbol{H} = \begin{bmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$L(\boldsymbol{q}) = \begin{bmatrix} q_w & q_x & q_y & q_w \\ q_x & q_w & -q_z & q_y \\ q_y & q_z & q_w & -q_x \\ q_z & -q_y & q_x & q_w \end{bmatrix}$$

$$\boldsymbol{Q} = \boldsymbol{H}^T \boldsymbol{T} L(\boldsymbol{q}) \boldsymbol{T} L(\boldsymbol{q}) \boldsymbol{H} \tag{2}$$

The constants and equations for calculating the derivative with respect to time of the dynamics:

$\boldsymbol{g}$ is the acceleration due to gravity in world coordinates:

$$\boldsymbol{g} = \begin{bmatrix} 0 \\ 0 \\ -9.81 \end{bmatrix} \frac{m}{s^2}$$

ℓ is the distance between the rotors:

$$\ell = 0.175\ m$$

k_t is the motor force constant:

$$k_t = 1.0$$

k_m is the motor torque constant:

$$k_m = 0.0245$$

$\boldsymbol{J}$ is the moment of inertia of tho quadrotor:

$$\boldsymbol{J} = \begin{bmatrix} 0.0023 & 0 & 0 \\ 0 & 0.0023 & 0 \\ 0 & 0 & 0.004 \end{bmatrix} kg \cdot m^2$$

m is the mass of the quadrotor:

$$m = 0.5\ kg$$

$$\dot{\boldsymbol{p}} = \boldsymbol{v} \tag{3}$$

$$\dot{\boldsymbol{q}} = \frac{1}{2} L(\boldsymbol{q}) \boldsymbol{H} \boldsymbol{\omega} \tag{4}$$

$$\hat{\boldsymbol{\omega}} = \begin{bmatrix} 0 & -\omega_z & \omega_y \\ \omega_z & 0 & -\omega_x \\ -\omega_y & \omega_x & 0 \end{bmatrix}$$

$$\dot{\boldsymbol{v}} = \boldsymbol{Q}^T \boldsymbol{g} + \frac{1}{m} \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ k_t & k_t & k_t & k_t \end{bmatrix} \boldsymbol{u} - \hat{\boldsymbol{\omega}} \boldsymbol{v} \tag{5}$$

$$\dot{\omega} = \boldsymbol{J}^{-1} \left(\begin{bmatrix} 0 & \ell k_t & 0 & -\ell k_t \\ -\ell k_t & 0 & \ell k_t & 0 \\ k_m & -k_m & k_m & -k_m \end{bmatrix} \boldsymbol{u} - \hat{\omega} \boldsymbol{J} \boldsymbol{\omega} \right) \tag{6}$$

$$\dot{\boldsymbol{x}} = \begin{bmatrix} \dot{\boldsymbol{p}} \\ \dot{\boldsymbol{q}} \\ \dot{\boldsymbol{v}} \\ \dot{\boldsymbol{\omega}} \end{bmatrix} \tag{7}$$

To get the state at the next time step, a fourth order Runge-Kutta integration function, f, is used:

$$\boldsymbol{x}_{n+1} = f(\boldsymbol{x}_n, \boldsymbol{u}_n) \tag{8}$$

All dynamics calculations are done in the "quadrotor_dynamics.jl" file in the GitHub repository.

IV. Red Quadrotor Trajectory

A time varying linear quadratic regulator (TVLQR) was used to create dynamically feasible trajectories for the red quadrotor. Four different reference trajectories, $\bar{X}$ were created.

- Constant velocity: the reference trajectory has a constant velocity of (0, -5, -5) meters per second from 0, 100, 100 to (0, 0, 0) meters.
- Zig zag z: use the constant velocity reference trajectory, but with a sine wave with amplitude of 6 meters per second and frequency of 0.2 hertz added to the z velocity.
- Zig zag y: use the constant velocity reference trajectory, but with a sine wave with amplitude of 5 meters per second and frequency of 0.2 hertz added to the y velocity.
- Zig zag x: use the constant velocity reference trajectory, but with a sine wave with amplitude of 5 meters per second and frequency of 0.2 hertz added to the x velocity.

For each reference trajectory the reference controls, $\bar{U}$, are all the controls to keep the quadrotor hovering. Choosing higher frequencies and amplitudes than the ones listed made the red quadrotor go out of control with the TVLQR controller.

For the TVLQR controller, the states quaternion part, $\boldsymbol{q}$, is converted to a three parameter Rodrigues vector, $\boldsymbol{r}$:

$$\boldsymbol{r} = \frac{1}{q_w} \begin{bmatrix} q_x \\ q_y \\ q_z \end{bmatrix} \tag{9}$$

The reduced state (with Rodrigues vector instead of quaternion) to use in the TVLQR controller is:

$$\tilde{\boldsymbol{x}} = \begin{bmatrix} \boldsymbol{p} \\ \boldsymbol{r} \\ \boldsymbol{v} \\ \boldsymbol{\omega} \end{bmatrix} \tag{10}$$

The difference between actual state, $\boldsymbol{x}_n$, and the reference state, $\bar{\boldsymbol{x}}_n$, is:

$$\Delta \boldsymbol{x}_n = \begin{bmatrix} \boldsymbol{p}_n - \bar{\boldsymbol{p}}_n \\ L(\bar{\boldsymbol{q}}_n)^T \boldsymbol{q}_n \\ \boldsymbol{v}_n - \bar{\boldsymbol{v}}_n \\ \boldsymbol{\omega}_n - \bar{\boldsymbol{\omega}}_n \end{bmatrix} \tag{11}$$

$$\begin{aligned} \boldsymbol{x}_{n+1} = \bar{\boldsymbol{x}}_{n+1} + \Delta \boldsymbol{x}_{n+1} &= f(\boldsymbol{x}_n, \boldsymbol{u}_n) \\ &= f(\bar{\boldsymbol{x}}_n + \Delta \boldsymbol{x}_n, \bar{\boldsymbol{u}}_n + \Delta \boldsymbol{u}_n) \\ &= f(\bar{\boldsymbol{x}}_n, \bar{\boldsymbol{u}}_n) + \boldsymbol{A}_n \Delta \boldsymbol{x}_n + \boldsymbol{B}_n \Delta \boldsymbol{u}_n \end{aligned} \tag{12}$$

The equivalent for the reduced states is:

$$\Delta \tilde{\boldsymbol{x}}_{n+1} = \tilde{\boldsymbol{A}}_n \Delta \boldsymbol{x}_n + \tilde{\boldsymbol{B}}_n \Delta \boldsymbol{u}_n \tag{13}$$

The objective function for the TVLQR is (the process to get $\tilde{\boldsymbol{A}}$ and $\tilde{\boldsymbol{B}}$ is explained in [4]):

$$\begin{aligned} \underset{\boldsymbol{x}, \boldsymbol{u}}{\text{minimize}} \quad & \sum_{n=1}^{N-1} [(\Delta \tilde{\boldsymbol{x}}_n)^T \boldsymbol{Q} (\Delta \tilde{\boldsymbol{x}}_n) + (\Delta \boldsymbol{u}_n)^T \boldsymbol{R} (\Delta \boldsymbol{u}_n)] \\ & + (\Delta \tilde{\boldsymbol{x}}_N)^T \boldsymbol{Q} (\Delta \tilde{\boldsymbol{x}}_N) \\ \text{subject to} \quad & \Delta \tilde{\boldsymbol{x}}_{n+1} = \tilde{\boldsymbol{A}}_n \Delta \boldsymbol{x}_n + \tilde{\boldsymbol{B}}_n \Delta \boldsymbol{u}_n \end{aligned} \tag{14}$$

Using the Riccati recursion equation, the gain matrix, $\boldsymbol{K}_n$ for each time step is calculated, starting from the last, $\boldsymbol{K}_{N-1}$:

$$\begin{aligned} \boldsymbol{P}_N &= \boldsymbol{Q} \\ \boldsymbol{K}_{N-1} &= (\boldsymbol{R} + (\tilde{\boldsymbol{B}})^T \boldsymbol{P}_N \tilde{\boldsymbol{B}}_{N-1})^{-1}((\tilde{\boldsymbol{B}}_{N-1})^T \boldsymbol{P}_N \tilde{\boldsymbol{A}}_{N-1}) \\ \boldsymbol{P}_{N-1} &= \boldsymbol{Q} + (\tilde{\boldsymbol{A}}_{N-1})^T \boldsymbol{P}_N \tilde{\boldsymbol{A}}_{N-1} - (\tilde{\boldsymbol{A}}_{N-1})^T \boldsymbol{P}_N \tilde{\boldsymbol{B}}_{N-1} \boldsymbol{K}_{N-1} \end{aligned} \tag{15}$$

With all the gain matrices calculated, the dynamically feasible trajectory can be determined with a forward roll out from the red quadrotors initial state:

$$\boldsymbol{u}_n = \bar{\boldsymbol{u}}_n + \boldsymbol{K}_n \Delta \tilde{\boldsymbol{x}}_n \tag{16}$$

$$\boldsymbol{x}_{n+1} = f(\boldsymbol{x}_n, \boldsymbol{u}_n) \tag{17}$$

The code to implement this section is in the files "quaternion.jl", "lqr.jl", and "red_traj.jl" in the GitHub repository.

V. Blue Quadrotor Interception State

The blue quadrotor will intercept the red one at a distance of 20 meters from the position it is defending, (0, 0, 0) meters. Using the red quadrotors current state, a straight line is drawn from its current position to (0, 0, 0). Then a point 20 meters from (0, 0, 0) is obtained.

$$\Delta \boldsymbol{p} = \boldsymbol{p}_{red} - \boldsymbol{p}_{defend} \tag{18}$$

$$\boldsymbol{p}_{intercept} = \frac{\Delta \boldsymbol{p}}{\|\Delta \boldsymbol{p}\|} 20 \tag{19}$$

The component of the red quadrotors velocity that is tangential to this line is multiplied by two seconds and then added to the intercept point calculated above. This lets the blue quadrotor look ahead instead of always being one time step behind the red quadrotor.

$$\boldsymbol{v}_{tang} = {}^W\boldsymbol{v}_{red} - \frac{\Delta \boldsymbol{p}}{\|\Delta \boldsymbol{p}\|}({}^W\boldsymbol{v}_{red} \cdot \frac{\Delta \boldsymbol{p}}{\|\Delta \boldsymbol{p}\|}) \tag{20}$$

$$\boldsymbol{p}_{goal} = \boldsymbol{p}_{intercept} + 2\boldsymbol{v}_{tang} \tag{21}$$

This goal position is passed into the blue quadrotors MPC.

VI. Blue Quadrotor MPC

The blue quadrotor is using an MPC with a horizon of 20 time steps to reach the goal state described in the section above. Each time step in the simulation the goal state is updated. The MPC uses the OSQP Julia package [5]. The objective of the MPC controller is:

$$\begin{aligned} \underset{\boldsymbol{z}}{\text{minimize}} \quad & \boldsymbol{z}^T \boldsymbol{P} \boldsymbol{z} + \boldsymbol{q}^T \boldsymbol{z} \\ \text{subject to} \quad & \boldsymbol{b}_l \le \boldsymbol{C}\boldsymbol{z} \le \boldsymbol{b}_u \end{aligned} \tag{22}$$

Where (here N is the MPC horizon length):

$$\boldsymbol{z} = \begin{bmatrix} \boldsymbol{u}_1 \\ \tilde{\boldsymbol{x}}_2 \\ \boldsymbol{u}_2 \\ \tilde{\boldsymbol{x}}_3 \\ \vdots \\ \boldsymbol{u}_{N-1} \\ \tilde{\boldsymbol{x}}_N \end{bmatrix}, \boldsymbol{P} = \begin{bmatrix} \boldsymbol{R} & & & & \\ & \boldsymbol{Q} & & & \\ & & \ddots & & \\ & & & \boldsymbol{R} & \\ & & & & \boldsymbol{Q} \end{bmatrix}, \boldsymbol{q} = \begin{bmatrix} -\boldsymbol{R}\boldsymbol{u}_{goal} \\ -\boldsymbol{Q}\tilde{\boldsymbol{x}}_{goal} \\ \vdots \\ -\boldsymbol{R}\boldsymbol{u}_{goal} \\ -\boldsymbol{Q}\tilde{\boldsymbol{x}}_{goal} \end{bmatrix}$$

$$\boldsymbol{c}_z = \begin{bmatrix} 0 & 0 & 1 & 0 & \dots & 0 \end{bmatrix}$$

$$\boldsymbol{C}_r = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 & \dots & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & \dots & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & \dots & 0 \end{bmatrix}$$

$$\boldsymbol{C}_t = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\boldsymbol{C} = \begin{bmatrix} \tilde{\boldsymbol{B}} & -\boldsymbol{I} & & & & & \\ & \tilde{\boldsymbol{A}} & \tilde{\boldsymbol{B}} & -\boldsymbol{I} & & & \\ & & & & \ddots & & \\ & & & & \tilde{\boldsymbol{A}} & \tilde{\boldsymbol{B}} & -\boldsymbol{I} \\ 0 & \boldsymbol{c}_z & & & & & \\ & 0 & 0 & \boldsymbol{c}_z & & & \\ & & & & \ddots & & \\ & & & & 0 & 0 & \boldsymbol{c}_z \\ 0 & \boldsymbol{C}_r & & & & & \\ & 0 & 0 & \boldsymbol{C}_r & & & \\ & & & & \ddots & & \\ & & & & 0 & 0 & \boldsymbol{C}_r \\ \boldsymbol{C}_t & 0 & & & & & \\ & 0 & \boldsymbol{C}_t & 0 & & & \\ & & & & \ddots & & \\ & & & & 0 & \boldsymbol{C}_t & 0 \end{bmatrix}$$

$$\boldsymbol{b}_l = \begin{bmatrix} \tilde{\boldsymbol{A}}\tilde{\boldsymbol{x}}_1 \\ 0 \\ \vdots \\ 0 \\ 0 \\ 0 \\ \vdots \\ 0 \\ -1 \\ -1 \\ \vdots \\ -1 \\ 0.3(mg/4) \\ 0.3(mg/4) \\ \vdots \\ 0.3(mg/4) \end{bmatrix}, \quad \boldsymbol{b}_u = \begin{bmatrix} \tilde{\boldsymbol{A}}\tilde{\boldsymbol{x}}_1 \\ 0 \\ \vdots \\ 0 \\ \infty \\ \infty \\ \vdots \\ \infty \\ 1 \\ 1 \\ \vdots \\ 1 \\ 2.5(mg/4) \\ 2.5(mg/4) \\ \vdots \\ 2.5(mg/4) \end{bmatrix}$$

$\boldsymbol{c}_z$ pulls out the p_z for each time step, and the lower bound of this parameter is 0 (i.e. constraint that the quadrotor doesn't go through the ground). $\boldsymbol{C}_r$ pulls out the Rodrigues vector of the reduced state. To keep yaw, pitch, and roll angles within ± 90 degrees each element of the Rodrigues vector must be between -1 and 1. $\boldsymbol{C}_t$ pulls out the control values, which are the thrust values for the quadrotor. Each rotor must have a thrust value between $0.3(m \cdot g/4)$ and $2.5(m \cdot g/4)$.

The OSQP solver is executed each time step to give a new control value, $\boldsymbol{u}_n$, which is then used to calculate the state at the next time step, $\boldsymbol{x}_{n+1}$. After each time step, the distance between the blue quadrotor and red quadrotor is calculated. In this simulation, an interception occurs if this distance is less than two meters.

The code to implement this is in the files "main.jl" and "mpc.jl" in the GitHub repository.

VII. Results

For each of the red trajectories, the blue quadrotor was able to intercept the red quadrotor. Here are the plots of for the red and blue trajectories for each of the different red trajectories used.

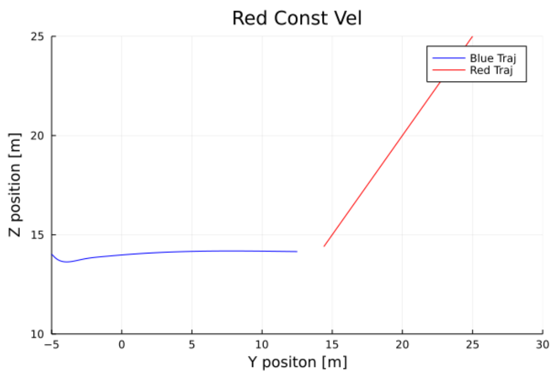

Fig. 1.
Y and Z positions of the red and blue quadrotor when red has a constant velocity trajectory

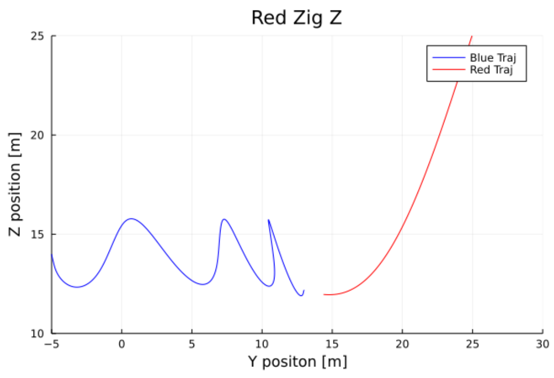

Fig. 2.
Y and Z positions of the red and blue quadrotor when red has an oscillating z velocity trajectory

To run the software that produced these results, follow the instructions in the README.md file in the GitHub repository.

VIII. Conclusion

This MPC worked well. Since both quadrotors had the same dynamics, the red quadrotor could not outmaneuver the blue quadrotor. To improve this MPC, a red quadrotor with better dynamic capabilities could be simulated. The algorithm to

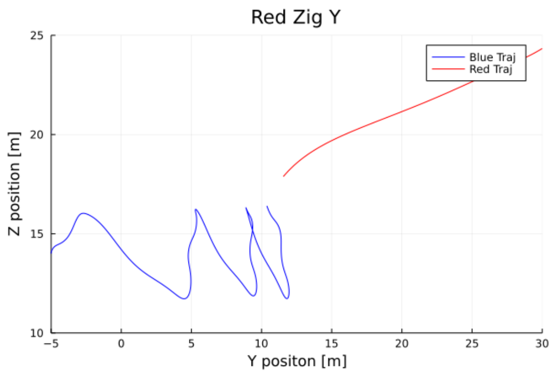

Fig. 3.
Y and Z positions of the red and blue quadrotor when red has an oscillating y velocity trajectory

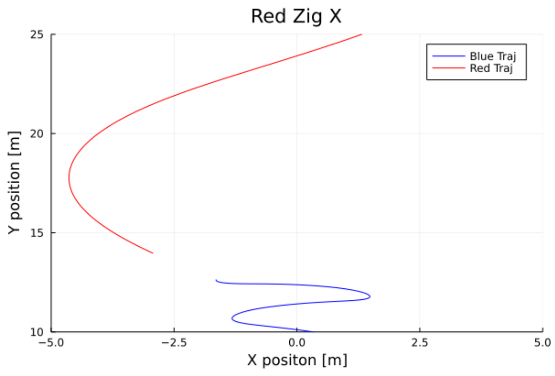

Fig. 4.
X and Y positions of the red and blue quadrotor when red has an oscillating x velocity trajectory

set the goal state for the blue quadrotor could be improved. Rather than using a straight line, the optimal dynamically feasible trajectory could be used instead. Noise could also be introduced into the blue quadrotor's measurement of the red's state. This would be more realistic since it is impossible to always get perfect measurements of the red quadrotor's state in a real life situation.

References

[1] K. Atherton. "Army tests using a grenade to stop drones with a net." armytimes.com https://www.armytimes.com/news/your-army/2019/02/11/army-tests-using-a-grenade-to-stop-drones-with-a-net/ (accessed May 10, 2022).

[2] N. Strout. "DARPA's newest system kills drones with stringy streamers." defensenews.com https://www.defensenews.com/unmanned/2021/06/08/darpas-newest-system-kills-drones-with-stringy-streamers/ (accessed May 10, 2022).

[3] Z. Manchester. (2022). Convex MPC [Notes and code]. Available: https://github.com/Optimal-Control-16-745/lecture-notebooks/tree/main/Lecture%209

[4] Z. Manchester. (2022). LQR with quaternions [Notes and code]. Available: https://github.com/Optimal-Control-16-745/lecture-notebooks/tree/main/Lecture%2015

[5] B. Stellato, G. Banjac, P. Goulart, A. Bemporad, S. Boyd. OSQP: an operator splitting solver for quadratic programs. Mathematical Programming Computation. Accessed: May 10, 2022. Available: https://osqp.org/docs/interfaces/julia.html