Graduate Project Portfolio - Nicholas Gurnard

Nicholas Gurnard
Robotics Masters Student at University of Pennsylvania – Thrives Working on Challenging Problems
Phone: +1 949 257 8760
Email: nk.gurnard@gmail.com || LinkedIn: Nicholas Gurnard || Github: ngurnard

***Purpose*—The purpose of this portfolio is to give any readers information on my professional endeavours and to get a feel for my personality. This portfolio is currently being updated and is designed to feature projects, accomplishments, and experiences I had in my graduate studies. This information is not to be distributed or replicated without my consent. If there are any questions or concerns, do not hesitate to contact me.**

Contents

I. Introduction

Hello! I'm Nicholas Gurnard, a Robotics Engineering student interested in controls, testing, modeling, and simulation for robotic systems. I am passionate about the intersection of hardware and software, and would love a role that has the opportunity to work on both in some capacity. Highly curious and am willing to learn and adapt quickly with efficient and purposeful communication.

As seen in this portfolio, I have broad coverage in topics such as trajectory/motion planning, computer vision, state estimation and filtering, controls, optimization, machine/deep learning, computer graphics, and more! NOTE: this portfolio is still being updated and takes time! Computer graphics and deep learning projects coming soon.

II. Autonomous VIO Drone Spring 2022

A. Introduction and System Overview

The objective of this project was to implement and tune a controller, waypoint generator, path planner, and visual state estimation algorithm (VIO - Visual Inertial Odometry) through a known map with a quadrotor in both simulation and on real hardware (Crazyflie quadrotor). The controller, planner, and estimation system were estimated within a custom python simulation environment encapsulating real Crazyflie model dynamics and then tested within a Vicon lab using the same maps that were tested in simulation.

For integration of our controller, trajectory planner, path planner, and VIO estimator with the Vicon space and Crazyflie, ROS (Robot Operating System) [1] was used to link all the devices and code together. The Crazyflie has several small reflective markers on its frame, which allow the Vicon camera system to track its position within the space to obtain the ground truth of the state in order to be compared against the VIO estimation algorithm and for tuning the controller. Before testing the VIO estimator, the Vicon system was used to stream the quadrotor's position data to a ROS node, which was used in the control loop that runs on the lab computer (efficitively using the ground truth so the quadrotor didnt crash during controller tuning because of the state estimator). The controller, trajectory planner, and path planner all run locally on the lab computer and communicate outputs to ROS, which then relays low level commands (motor speeds and thrust) to the Crazyflie's onboard controller.

B. Controller

The controller used on the quadrotor was a geometric nonlinear controller. The block diagram of the nested control loop is shown in Fig 1. The coordinate system of the quadrotor is shown in Fig 2. The controller is based on the geometric intuition that we point the quadrotor's b3 axis in the desired direction and direction of applied thrust generated by the motors.

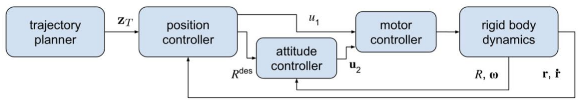


Figure 1: The Position and Attitude Control Loop [2]

.

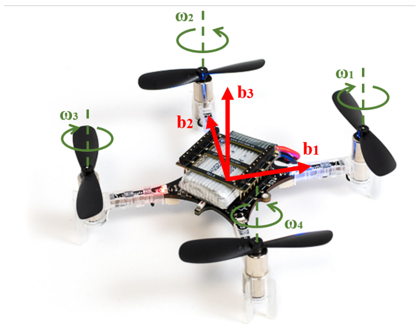


Figure 2: Quadrotor Coordinate System

.

More specifically, the controller is built with an outer loop and an inner loop. The outer loop is a PD position controller which can be described with

$$\ddot{r}^{des} = \ddot{r}_T - K_d(\dot{r} - \dot{r}_T) - K_p(r - r_T) \tag{1}$$

Where, r is the current position vector of the quadrotor in the world frame, r_T is the desired position vector of the quadrotor in the world frame. From Eq. 1, the combined thrust (u_1) of the four motors can be calculated with

$$u_1 = b_3^T F^{des} = b_3^T(m\ddot{r}^{des} + \begin{bmatrix}0 & 0 & mg\end{bmatrix}^T) \tag{2}$$

Where, b_3^T is the coordinate vector of the quadrotor's b_3 axis with respect to the world frame, m is the weight of the quadrotor, and g is the gravity factor.

The inner loop is a PD attitude controller described by

$$u_2 = I(-K_R e_R - K_\omega e_\omega) \tag{3}$$

Where, u_2 is formed by the torque generated by each motor, I is quadrotor's inertia, e_R is the error between the current orientation and the desired orientation, e_ω is the error between the current angular velocity and the desired angular velocity. e_R can be calculate with

$$e_R = \frac{1}{2}(R^{des^T} R - R^T R^{des})^\vee \tag{4}$$

Where, R is the current rotation matrix,$R^{des} = \begin{bmatrix}1{b_2}^{des} \times {b_3}^{des} & {b_2}^{des} & {b_3}^{des}\end{bmatrix}^T$is the desired rotation matrix. The $^\vee$ operator is the inverse of the hat operator, which converts a 3x3 matrix into a 3x1 vector, as follows.

$$\widehat{\boldsymbol{\omega}} = \begin{bmatrix} 0 & -\omega_3 & \omega_2 \\ \omega_3 & 0 & -\omega_1 \\ -\omega_2 & \omega_1 & 0 \end{bmatrix}$$
$$\boldsymbol{\omega} \times \boldsymbol{b} = [\widehat{\boldsymbol{\omega}}][\boldsymbol{b}]$$

In the equation 1 and 3, the K_d, K_p, K_R, K_ω are diagonal positive definite gain matrices. Approximate values for a 2 part phycial lab experimanet are as follows: // For the experiment part one:

$k_p = \begin{bmatrix}3.36 & 3.36 & 3.78\end{bmatrix}$, unit s^{-2};
$k_d = \begin{bmatrix}5.04 & 5.04 & 3.5\end{bmatrix}$, unit s^{-1};
$k_r = \begin{bmatrix}210.7 & 210.7 & 92.4\end{bmatrix}$, unit s^{-2};
$k_\omega = \begin{bmatrix}14 & 14 & 10.5\end{bmatrix}$, unit s^{-1};

For the experiment part two:

$k_p = \begin{bmatrix}5 & 5 & 5\end{bmatrix}$, unit s^{-2};
$k_d = \begin{bmatrix}3.5 & 3.5 & 3.5\end{bmatrix}$, unit s^{-1};
$k_r = \begin{bmatrix}3750 & 3750 & 135\end{bmatrix}$, unit s^{-2};
$k_\omega = \begin{bmatrix}100 & 100 & 20\end{bmatrix}$, unit s^{-1};

In this experiment, k_p and k_d are used to control the position of the quadrotor. More specifically: k_p is the proportional gain, which directly affects the position error and can increase the system error-reaction speed and accuracy. k_d is the derivative gain, which directly affects the linear speed error. In practice, we increased the P gain until there is overshooting, and then increased the D gain to reduce (or eliminate) the overshooting. The P and D gains were then incrementally increased until the response time was satisfactory. k_r and k_ω are used to control the orientation of the quadrotor. More specifically: k_r works similar to k_p, directly affecting the orientation error and k_ω works similar to k_d, directly affecting the angular velocity error.

When controlling the quadrotor, the control command commands the motor speeds. However, from the position controller we can only get the four-motors' combined thrust (u_1), and from the attitude controller we can only get the torque of each motor (u_2). Therefore, none of them can be directly used as a command for the quadrotor's motors. In order to control the quadrotor's motion as what we expected, we need to later on use the thrust and torque to calculate and recover each motor's speed and use that to command the quadrotor.

From the experiment, it was noticed that there are several possible factors which can cause differences between the experiment and the simulation. First is the weight of the quadrotor: from Eq. 2 we know that the weight of the quadrotor will directly influence the computed thrust. Therefore, the difference between the simulation and reality weight will cause a b_3 direction position shift and thus we may need to adjust the quadrotor's weight in the simulation code to achieve a better performance. Additionally, the uneven distribution of the quadrotor mass may cause drifting to the direction with the higher weight. Second is the motor saturation: in the simulation the performance of the motor is always ideal, however in the reality the motors cannot produce as much speed as we want due to the motor saturation. Therefore, we adjusted the control gains to achieve a better performance. Note: in simulation the motor saturation was accounted for, however it did not account for the nonlinear speed trend seen in real motors.

The performance of the position controller is shown in Figure 3.

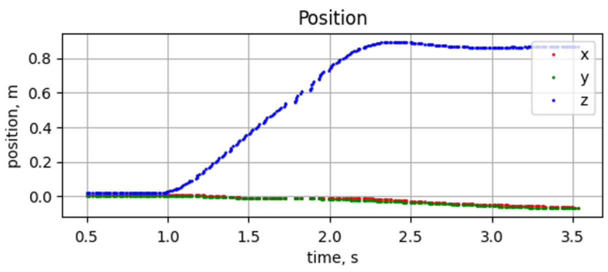


Figure 3: Position vs Time step response plot

From Figure 3 it is known that the overshoot for this control curve is near $\frac{0.06}{0.8} = 7.5\%$. This number was brought down to approximately 4.5% was some tuning effort. In simulation, the overshoot was 3.9%.

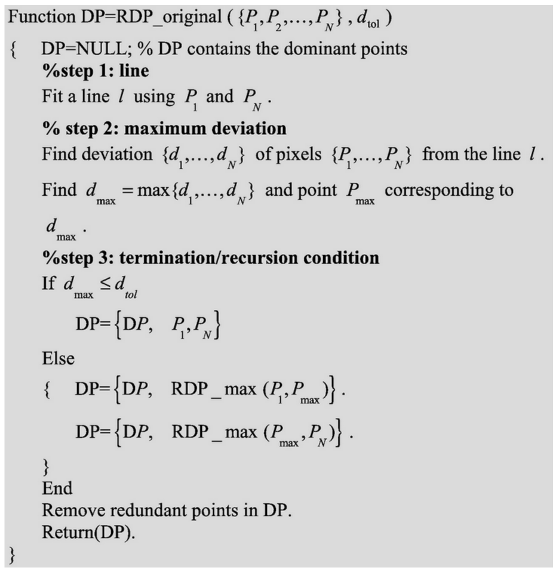
$$overshoot = e^{-\xi\pi/\sqrt{1-\xi^2}} * 100\% \tag{5}$$

The approximate damping ratio was calculated to be approximately 0.7 (same as simulation). From the curve we can read that: the steady state error is about $0.85 - 0.80 = 0.05m$, which may be caused by the quadrotor weight difference between the simulation and the experiment discussed earlier. The simulation SS error was approximately 0. The rise time is about $1.1s$ ($1.1s$ simulation!) and settling time is about $1.6s$ ($2s$ simulation!).

The orientation performance (plot not shown) of the quadrotor showed slightly higher overshoot ($10 - 12\%$), however with significantly less rise time (0.015s) and settling time (0.6s), which is expected of the attitude controller since a much more aggressive controller is necessary for good flight performance. Simulation showed an overshoot of 12.5%, a rise time of 0.01s, and a settling time of 0.5s. The performance of this controller was determined reasonable and relatively good for a high speed environment with average obstacle density. In a high density environment, a slightly better performing quadrotor would be needed (note: this is likely due to a hardware limitation - the tuning was comparatively very good for what we had).

C. Trajectory Generator

A map of a known environment was first discretized into voxels of a user specified resolution. This map was then converted from the workspace $W \subset \mathbb{R}^3$ to the configuration space $C \subset \mathbb{R}^3$ such that the drone could be abstracted into a point, allowing the optimal path to be computed using standard graph search algorithms. Workspace obstacles were represented in the C-space by inflating the obstacle using the known geometry of the quadrotor plus some margin. The optimal path from a specified start point to goal point was computed within the free-space of the C-space using the A^* algorithm. The chosen heuristic used within the A^* algorithm was the Euclidean Distance in $\mathbb{R}^3$ since it is always admissible and consistent in this context.



```
Function DP=RDP_original ( $\{P_1,P_2,\ldots,P_N\}$ , $d_{tol}$ )
{   DP=NULL; % DP contains the dominant points
    %step 1: line
    Fit a line $l$ using $P_1$ and $P_N$ .
    % step 2: maximum deviation
    Find deviation $\{d_1,\ldots,d_N\}$ of pixels $\{P_1,\ldots,P_N\}$ from the line $l$ .
    Find $d_{max} = \max\{d_1,\ldots,d_N\}$ and point $P_{max}$ corresponding to
    $d_{max}$ .
    %step 3: termination/recursion condition
    If $d_{max} \le d_{tol}$
        DP= $\{DP,\ P_1,P_N\}$
    Else
    {   DP= $\{DP,\ RDP\_max\ (P_1,P_{max})\}$ .
        DP= $\{DP,\ RDP\_max\ (P_{max},P_N)\}$ .
    }
    End
    Remove redundant points in DP.
    Return(DP).
}
```

Figure 4: Ramer-Douglas-Peuker algorithm pseudocode [4]

Given the set of waypoints generated by the A^* graph search algorithm, the Ramer-Douglas-Peuker (RDP) [3] algorithm was used to prune the path so that points that were colinear or near colinear were eliminated from the path, albeit the endpoints of the line. RDP works recursively as shown in Figure 4.

RDP often prunes the path too much or not enough since it is mainly used to prune colinear or near colinear points. The path was further pruned by eliminating points that were too close together given a certain tunable minimum threshold. Each voxel had 26 neighboring voxels, including diagonals, and thus the minimum threshold was approximately the size of 2 times the diagonal voxel distance to a neighboring voxel. If the path became over-pruned from RDP and eliminating close voxels, waypoints were injected back into the path if 2 consecutive voxels were at a distance away from each other that is greater than a tunable user defined maximum threshold.

The importance of pruning the planned path well is because of the time allocation in the trajectory generation. The quadrotor flies with nonlinear dynamics, and thus time allocation between waypoints is also nonlinear. The function to compute the time allocation is not known, and thus optimally pruning the waypoints is critical for a smooth, safe, and achievable trajectory. With an optimall pruned path, the time allocation between waypoints was computed as follows:

1) Specify an average trajectory velocity
2) Compute Euclidean Distance between waypoints
3) Compute time between waypoints as:
   $T = \frac{distance^{1/3}}{v_{avg}}$

Depending on the map geometry, it is sometimes better to use a fraction of $\frac{1}{2}$ instead of $\frac{1}{3}$ in the exponent of distance in step 3. It is also critical to allow time for acceleration and deceleration at the beginning and end of the trajectory. The time for the first and last segment were multiplied by a factor of 2 and 1.5 respectively. Without these, the trajectory was too aggressive or the drone would hit a wall. With proper time allocation and path pruning, the smoother trajectory meant the VIO alogithm could track features better and get a better estimate of its current state.

The RDP algorithm was DRASTICALLY improved with the implementation of a ray tracing algorithm. Given the set of dense waypoints from the path planner, instead of pruning with RDP, the ray tracing algorithm solved the issues related to pruning too much or too little, and also improved upon the difficulty of tuning the time allocation. The ray tracing algorithm works by looping through every point in the path starting at the first point. For each future point after the current point in the loop, draw a vector between the current point and all future points. The future point that is farthest from the current point while not having any obstacle along the constructed is kept (meaning there is no collision), all other points between are pruned. Then the algorithm continues until it moves through the entire path. Figure 5 shows an illustration of the algorithm for a single point early in the trajectory. This pruning algorithm allowed for long segments in which the drone can move quickly, but also smoothly.

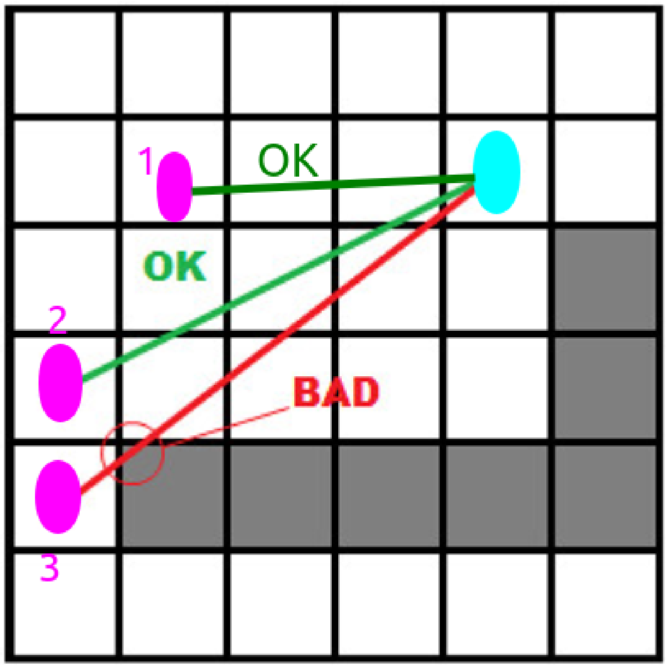


Figure 5: Ray tracing (line-of-sight) pruning algorithm. Pink = potential future points.

To prove the smoothness and effectiveness of the planned trajectory with the ray tracing pruning algorithm, plots of the acceleration and jerk were compared. The comparisons are shown in Figures 6, 7, and 8. It can be seen that the ray tracing algorithm created the smoothest acceration and jerk curves.

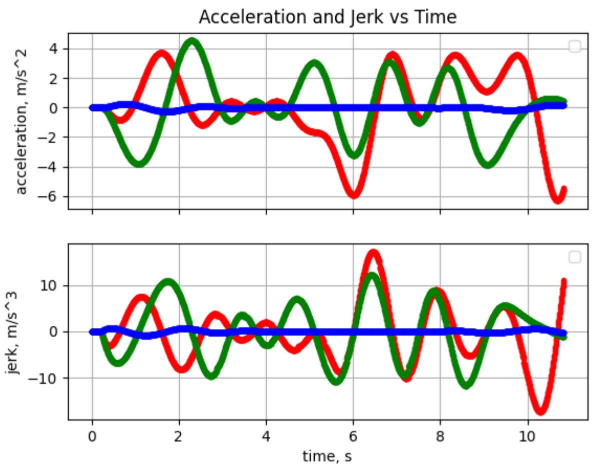


Figure 7: RDP Pruning

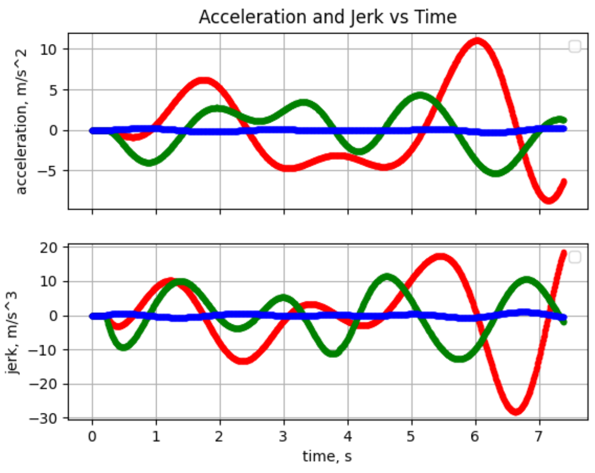


Figure 8: Ray Tracing Pruning

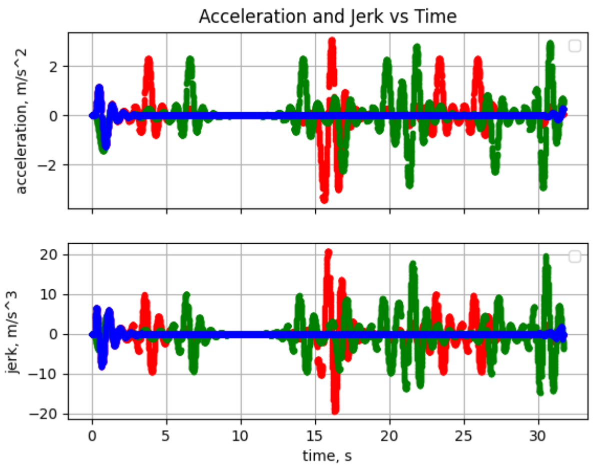


Figure 6: No Pruning

Given the time between waypoints, a minimum snap trajectory was implemented. The optimal minimum snap trajectory was computed with

$$x^*(t) = \underset{x(t)}{\arg\min} \int_0^T \mathcal{L}(x^{(4)}, \dddot{x}, \ddot{x}, \dot{x}, x, t)dt \tag{6}$$

Where $\mathcal{L}(x^{(4)}, \dddot{x}, \ddot{x}, \dot{x}, x, t)$ is the running cost for snap (fourth derivative; $n = 4$). This minimization can be solved with the Euler Lagrange equation since it is a necessary condition satisfied by $x^*(t)$.

$$\frac{\partial \mathcal{L}}{\partial x} + \frac{d}{dt}(\frac{\partial \mathcal{L}}{\partial \dot{x}})(-1)^n + \cdots + \frac{d^n}{dt^n}(\frac{\partial \mathcal{L}}{\partial x^{(n)}})(-1)^n = 0$$

$$\frac{\partial \mathcal{L}}{\partial x} - \frac{d}{dt}(\frac{\partial \mathcal{L}}{\partial \dot{x}}) + \frac{d^2}{dt^2}(\frac{\partial \mathcal{L}}{\partial \ddot{x}}) - \frac{d^3}{dt^3}(\frac{\partial \mathcal{L}}{\partial \dddot{x}}) + \frac{d^4}{dt^4}(\frac{\partial \mathcal{L}}{\partial x^{(n)}}) = x^{(8)} = 0$$

$$x = c_0 + c_1 t + c_2 t^2 + c_3 t^3 + c_4 t^4 + c_5 t^5 + c_6 t^6 + c_7 t^7 \tag{7}$$

Equation 7 is a polynomial of degree $(2n - 1)$ with 8 unknowns. A trajectory of m segments can therefore have $8m$ total unknowns that must be computed. The $8m$ unknowns, $\bar{c}$ can be computed by solving $A\bar{c} = b$, where $A \in \mathbb{R}^{8mx8m}$, $\bar{c} \in \mathbb{R}^{8mx1}$, and $b \in \mathbb{R}^{8mx1}$. b can be formulated by taking the derivative of equation 7. The start and end points of the full trajectory can have constraints up to degree $x^{(n-1)} = x^{(3)}$, which satisfies 8 of the $8m$ necessary constraints.

$$\dddot{x}_{start,end}, \ddot{x}_{start,end}, \dot{x}_{start,end}, x_{start,end} = 0$$

For intermediate waypoints, continuity constraints were defined up to the $2(n-1)^{th}$ derivative, satisfying the remaining constraints. The time at the beginning of each segment is considered as time 0, and the time at the end of each segment is t_{m_i}. The following is an example of constraints between segment 1 and segment 2:

$$x_1(t_1) = x_1;\ x_2(0) = x_1$$
$$\dot{x_1}(t_1) = \dot{x_2}(0);\ \ddot{x_1}(t_1) = \ddot{x_2}(0)$$
$$\dddot{x_1}(t_1) = \dddot{x_2}(0);\ x_1^{(4)}(t_1) = x_2^{(4)}(0)$$
$$x_1^{(5)}(t_1) = x_2^{(5)}(0);\ x_1^{(6)}(t_1) = x_2^{(6)}(0)$$

Given all endpoint and continuity constraints, $A\bar{c} = b$ can now be solved explicitly for $\bar{c}$. The following link provides an example of $A\bar{c} = b$ given only 2 segments: LINK HERE. The following link provides an example of $A\bar{c} = b$ given only 3 segments: LINK HERE.

Given a time t, now that all constraints $\bar{c}$ are solved, a desired flat output and its derivatives can be computed for the entire trajectory using Equation 7 and its derivatives. This flat output is then mapped to states and inputs in order to control the quadcopter with minimum snap.

A plot of quadcopter position error and velocity are shown in Figures 15 and 16. The planned trajectory was highly smooth and feasible, especially at lower speeds. At higher speeds, the trajectory became less smooth/feasible since the planned trajectory demanded more aggressive maneuvers, discussed more in section II-E.

D. VIO Based State Estimator (ESKF) and Yaw Control

The state of the quadrotor was estimated using VIO instead of the ground truth with the Vicon, thus, aggressive maneuvers would result in poor state estimates. This was implemented with an Error-State Kalman Filter (Complementary filter was used in the experiment due to the faster compute speed) where the input data was stereo correspondence data and IMU onboard data (gyroscope and accelerometer). The stereo correspondences were computed with RANSAC and least squares solving. In the ESKF, aggressive maneuvers would result in higher noise in the IMU since the acceleration, which is more noisy with aggressive maneuvers, must be integrated to get the velocity and position thus further propagating the error. Additionally, the VIO algorithm uses stereo images to extract and track features. With aggressive maneuvers, feature tracking and feature matching becomes more noisy, resulting in dropped features, and thus the measurement from the camera becomes less trustworthy in the ESKF. The error state covariance matrix would then reflect the extra noise in the poor vision measurement, and thus the Kalman gains would reflect this poor measurement. This would potentially result in your filter trusting the the wrong (more noisy) sensor such as the IMU over the vision measurement. Additionally, having too aggressive of a controller can saturate the motors, meaning the drone will not be able to maneuver as aggressively as demanded, and may result in a crash.

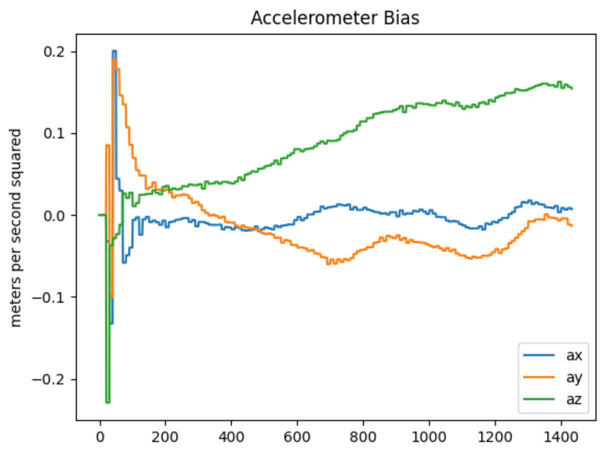


Figure 9: Accelerometer bias without initial hovering

Throughout the trajectory, the drone maintained the same yaw heading without yaw control. In the case where the trajectory was not along the optical axis of the stereo camera, the drone was effectively tracking features that were to the side of it rather than in front of it. Without yaw control on a physical drone, features not along the optical axis, which usually points forward on the drone's body frame, are not informative and are likely to be dropped (more feature correspondences = better state estimate in general) when using a feature tracking algorithm such as Lukas-Kanade optical flow (in simulation, feature tracking/correspondence is perfect and trivial). Track-

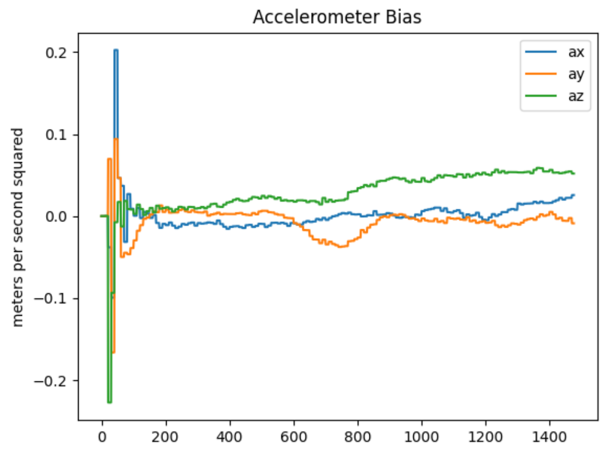


Figure 10: Accelerometer bias with initial hovering

ing features in real life is much more difficult when they are estimated and the drone is moving quickly. Therefore, yaw control was implemented by setting $yaw\_control = True$ in `world_traj.py`. The velocity vector of the flat output at each timestep was projected onto the x-y plane in order to find the yaw angle relative to the initial orientation using the euqation $\text{proj}_{\vec{N}}^{\vec{k}} = (\vec{k} \cdot \vec{N})/||\vec{N}||^2 * \vec{N}$. The veclocity vector was first projected onto the vector $\vec{N}$ which is normal to the x-y plane, then the projected vector on the x-y plane was computed using vector addition: $\text{proj}_{XY}^{\vec{k}} = \vec{k} - \text{proj}_{\vec{N}}^{\vec{k}}$. The yaw heading angle was then computed using inner-product of the projected vector and the y-axis (initial optical axis direction): $\theta = \arccos(\vec{y} \cdot \text{proj}_{XY}^{\vec{k}})$. Since $R(\arccos) = [0, \pi]$, θ was multiplied with $\text{sign}(\vec{y} \times \text{proj}_{XY}^{\vec{k}})$ in order to cover the range $[0, 2\pi]$. At angles of $0, \pi$, the quad experienced yaw angle wrap around sometimes making the drone spin $180°$, which can be fixed with some post processing (ran out of time).

Because the error state covariance matrix and gyroscope/accelerometer biases are difficult to know in reality, they are initialized to random values in practice. Because of this, the initial covariance and bias values are likely far from the truth at the start of the trajectory. Beginning to fly while the estimated biases and covariance are poor would result in an extremely poor state estimate, and thus the drone would struggle to localize and accurately track the desired trajectory. Making the drone to hover for a short period, say between $\frac{1}{4}$ to 1 second, allowed for the bias values and covariance to converge before flight by getting readings from the IMU and VIO with minimal noise and accurate feature tracking. Having a more accurate covariance matrix, which directly influences the Kalman gain in the ESKF, makes it such that the quad trust the sensor with more accurate readings. Implementing a brief hover period resulted in more successful tracking of the trajectory. To visualize the effectiveness of initially hovering, Figures 9 and 10 shows how the bias estimate is more stable, i.e. does not drift from the true bias as much, throughout the trajectory.

E. Maze Flight Experiments

The objective of this section is to give a review of the results obtained in the experiments. Figures 11, 12, 13 show 3D plots of the obstacles, waypoints, the planned trajectory, and the actual flight of each enviroment experimented on. It is clear that for maze 1 and maze 2 the actual flight is close to the simulated one. In the case of maze 3, the difference is much more significant.

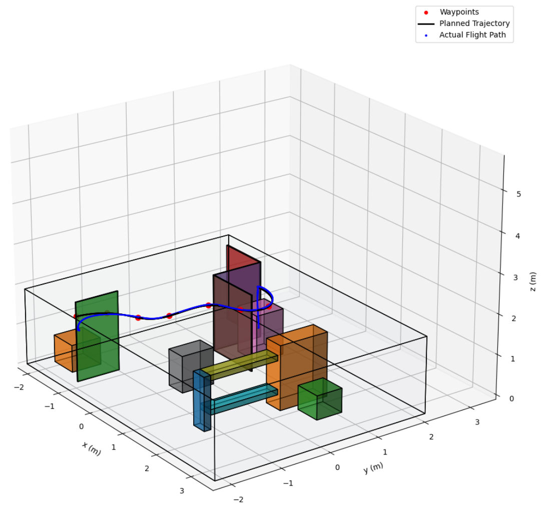


Figure 11: Obstacles, waypoints, planed trajectory, actual flight: Map 1

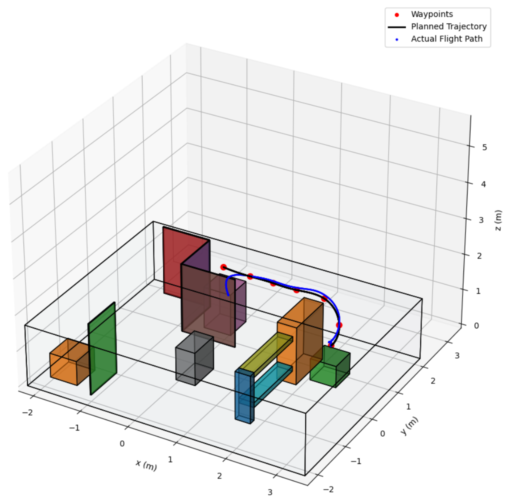


Figure 12: Obstacles, waypoints, planed trajectory, actual flight: Map 2

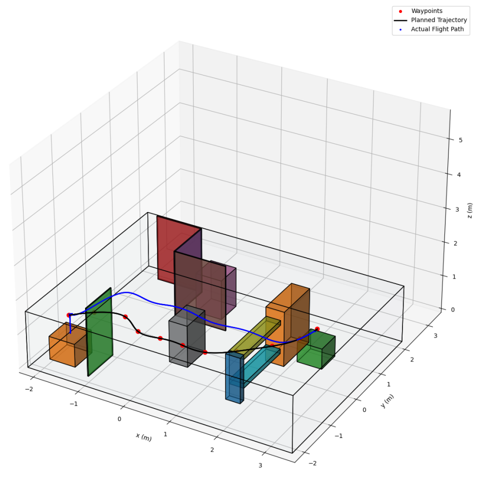


Figure 13: Obstacles, waypoints, planed trajectory, actual flight: Map 3. Note: the planned trajectory had a lower ceiling in simulation than in experimentation, and thus the actual flight path went over an obstacle instead of around because that was more optimal.

As mentioned, we were able to run the experiments for all maps at 1, 1.5, and 1.8 m/s speeds. Figure 14 presents the position and velocity vs. time for map1 at 1.5 m/s. The position plot presents a smooth trajectory, which is exactly what was desired with the minimum snap trajectory. The velocity of the quadrotor changed more aggressivley, however the velocity curves are still smooth and controlled, and do not exhibit any indication of wobble during flight (success!).

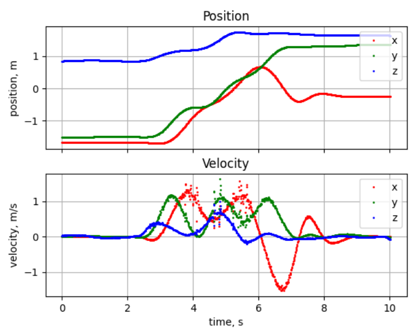


Figure 14: Position and Velocity over time for map 1 at 1.5 m/s.

To analyze tracking error, the position/velocity from simulation data for maze1 at a speed of 1 m/s was compared to the actual position/velocity obtained in the experiment. The simulation used a sampling rate 5x that of the Vicon records, so this had to be accounted for when overlaying. Additionally, the plots were manually aligned in time. From Figures 15 it can be observed that the approximate position error between the simulation and the experimental trajectory are 3.66% for x, 2.33% for y, and -22.74% for z. The error in z is large because of the weight differential between the experiment and simulation discussed earlier. The actual weight used in the experiment was 1.175 * drone_weight, reducing the tracking error considerably to approximately 2%. Since the tracking error was excellent at 1m/s, the speed was incremented to 1.5m/s and 1.8m/s.

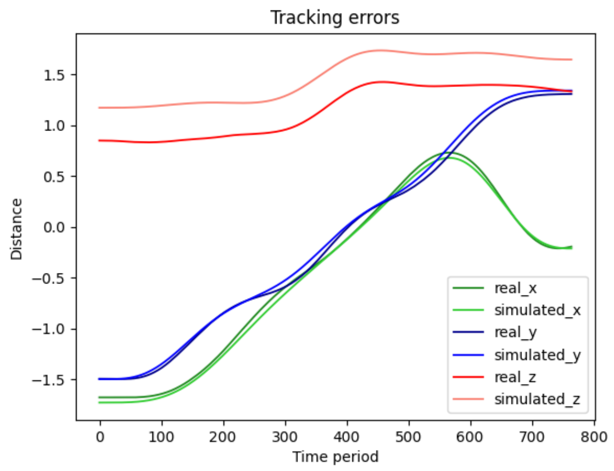


Figure 15: Position Error Tracking

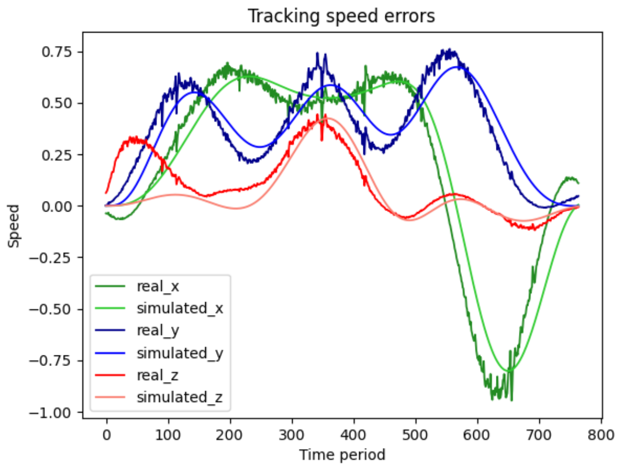
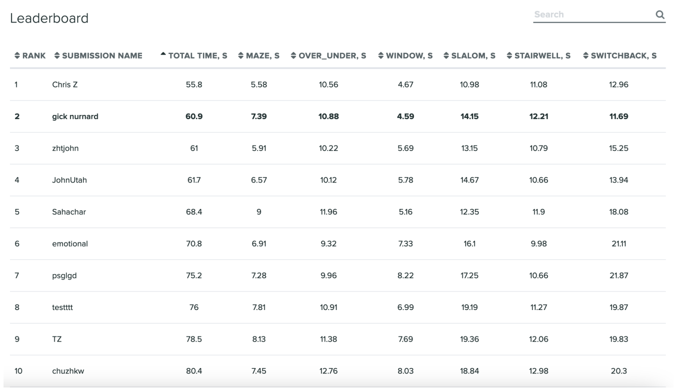


Figure 16: Velocity Error Tracking

F. Conclusion

This project was a huge success and I learned an incredible amount about the sim-to-real gap, state estimation, controller design and tuning, path planning, trajectory optimization, and more. In a competition of approximately 100 students, my experimentation team (Team of 4 - Jessica Yin, Luis Escobar, Tianyun Zhao, and me) had the best experimental drone flight and was able to perform the most aggressive maneuvers and complete the planned trajectories the fastest. In simulation, my personal (solo - no team) code placed 2nd in the competition across a series of maps, shown in the Figure 17 (Note - my name was Gick Nurnard which is just a play on my name by swapping the first letters of my first and last name).



Leaderboard

Search

RANK	SUBMISSION NAME	TOTAL TIME, S	MAZE, S	OVER_UNDER, S	WINDOW, S	SLALOM, S	STAIRWELL, S	SWITCHBACK, S
1	Chris Z	55.8	5.58	10.56	4.67	10.98	11.08	12.96
2	**gick nurnard**	**60.9**	**7.39**	**10.88**	**4.59**	**14.15**	**12.21**	**11.69**
3	zhtjohn	61	5.91	10.22	5.69	13.15	10.79	15.25
4	JohnUtah	61.7	6.57	10.12	5.78	14.67	10.66	13.94
5	Sahachar	68.4	9	11.96	5.16	12.35	11.9	18.08
6	emotional	70.8	6.91	9.32	7.33	16.1	9.98	21.11
7	psglgd	75.2	7.28	9.96	8.22	17.25	10.66	21.87
8	testttt	76	7.81	10.91	6.99	19.19	11.27	19.87
9	TZ	78.5	8.13	11.38	7.69	19.36	12.06	19.83
10	chuzhkw	80.4	7.45	12.76	8.03	18.84	12.98	20.3

Figure 17: Placed 2nd in racing competition of 100 competitors

G. *Videos and Github*

Videos of the drone flying in the experiments can be found at this video link: DRONE VIDEO LINK. The github repository for this project can be found here: GITHUB LINK.

III. F1TENTH Autonomous Racing Spring 2023

The F1TENTH Autonomous Racing Challenge uses an outfitted Traxxas car equipped with a Jetson Nano to autonomously race up to 40mph. This project included algorithm development in navigation and path/trajectory planning, motion planning, optimization for MPC and racelines, computer vision and deep learning, controls, and more. The repositories for all of the projects can be found on my github, each repo having the prefix "f1tenth_" and then the name of the algorithm developed. Each repository consists of a ROS2 foxy node that can be cloned directly onto the F1TENTH platform and be ready to race. The repository "f1tenth_interfaces" is required for pure pursuit nodes because it has a custom message for the waypoints. Please find my GitHub HERE. The following video shows a lot of clips during the F1TENTH Journey: LINK. Videos of the car in action can be found in the SUBMISSION.md of each of the repositories (PS - it is really cool, go check it out!).

The final project for the course was pitch control in mid-air, as described in the following write-up: LINK

IV. Robotic Arm - Franka-Emika Panda Spring 2023

This project was a part of a class that focused on the fundamental kinematic, dynamic, and computational principles underlying most modern robotic systems. The main topics of the course included: rotation matrices, homogeneous transformations, manipulator forward kinematics, manipulator inverse kinematics, Jacobians, path and trajectory planning, sensing and actuation, and feedback control. The material is reinforced with hands-on lab exercises involving a robotic arm (Franka-Emika Panda) after performing simulation in a Gazebo environment. The final project of the course was an entirely self-guided pick-and-place competition that was intentionally left open-ended.

For the sake of preventing this document from getting too large, the following links will redirect to reports written throughout the course:

- Forward Kinematics (FK) Lab
- Velocity Inverse Kinematics (IK) Lab
- Optimization Based IK Lab
- Path and Trajectory Planning Lab - Artificial Potential Field and RRT*
- Final Pick-and-Place Head-to-Head Competition

V. Ensemble Filtering - State Estimation Spring 2022

A. *Abstract*

In time critical systems, it is essential that the state estimation pipeline is quick and accurate. Current methods often struggle to achieve both. In this project we attempt to rectify this problem by constructing an ensemble filtering method by combining the state estimates from multiple filters to get a better overall estimate. The ensemble consists of fast, weak learners that can compete with an accurate, but slow filtering algorithm. There are 3 methods to form the ensemble: a simple average, a perceptron network, and a dense neural network with loss functions of MSE loss and a custom loss function. We demonstrate that the performance of the ensemble, in specific the dense neural network, is better than the individual performance of each filter tested on the EuRoC dataset.

B. *Introduction*

State estimation of a robot is critical for any robotics pipeline, where the Kalman filter variants are the backbone of most modern state estimation algorithms. However, the dynamics of a system are often hard to model, and thus a single Kalman filter variant may not be able to capture all of the intricacies of the dynamics. There are many variations of the Kalman filter that attempt to reduce assumptions about the system dynamics, for example the Unscented Kalman Filter (UKF) and Error State Kalman Filter (ESKF), however still rely on a simplification of the true model dynamics [5]. Previous work has introduced the idea of using multiple Kalman filters together in order to limit model simplification and assumptions [6] [7] [8]. Additionally, in a system that experiences rapid changes in its state, for example aggressive drone flight, the Kalman filter can be slow in recovering the true state once there is a major deviation [5].

When tasked with a real system, the speed of the filter is often a bottleneck that causes the system to fail [9] [10]. However, these same aggressive systems also rely on excellent state estimation to avoid failure. Fast, but less accurate, filtering algorithms are often the replacement for slow, accurate algorithms [10]. In this project, we introduce the idea of combining fast and less accurate filters together to get a better state estimate while still saving on computation time.

The proposed algorithm has 2 main applications: time critical systems that require an ensemble of fast but less accurate filters, and delay tolerant systems that would allow for an ensemble of several slow but extremely accurate filters. This project borrows the ensemble learning idea from field of machine learning, where several weak learners are combined together in an ensemble to create a powerful unbiased estimator such as random forests, which can then be passed through a neural network to capture the nonlinearity of the dynamics [11] [12].

1) Contributions: We demonstrate that the performance of the ensemble is better than the performance of each individual filter for system state estimation. We present 3 methods to combine the filters, namely simple averaging, a single layer perceptron, and a neural network with loss functions of MSE loss and custom loss function.

C. Background

Kalman filters are a family of algorithms that use a series of sensor measurements observed over time and produce state estimates for the system. The Kalman filter and its variants broadly work in 2 stages, namely propagation step and update step. In the propagation step, we use the estimate of the state at time k and estimate the state at time $k+1$ before the observation arrives. Once the observation is received, the update step is executed and the estimate of the state for time $k+1$ is refined. This process is repeated recursively for each new observation received.

If the system has linear dynamics, linear observation models, and Gaussian noise in the sensor readings, the Kalman filter is the most efficient state estimator. But as in most practical systems, the system dynamics and/or the observation models are nonlinear which lead to degraded performance on the vanilla Kalman filters. There are many variations like UKF and Extended Kalman Filters (EKF) which approximate the nonlinear dynamics of the system to a linear function and then use the propagate and update equations from the Kalman filter algorithm. Since these variations approximate the nonlinearity in the system dynamics, they are often noisy in their estimates. Different filters use different techniques such has Taylor series approximation, sigma points transformation etc. for the non-linear to linear approximation.

The inspiration for ensemble systems comes from the idea of random forests. In random forest algorithm, multiple decision trees are combined to produce a prediction output. Each decision tree in the ensemble is a weak learner but their combination produces an efficient result that is unbiased. This idea can be extended to filtering systems as well. Modern robotics systems require fast and accurate filtering system for efficient state estimation and trajectory tracking. An accurate but slow filter and a fast but inaccurate filter can lead to poor trajectory tracking and produce drifts in the robots movement. Thus, in this project we develop an ensemble based filtering system that uses multiple fast filters to produce an accurate estimate of the system's state.

D. Related Work

Multi stage filtering is an effective way to combine measurements from various sensors. This idea is demonstrated in [9] [13] [14]. [9] demonstrates a 3 stage system to combine the sensor measurements from a stereo camera and IMU. It's implementation follows that of [13] where the IMU sensor data is used to propagate the state of the system and when the camera measurements are received the state update step is performed. [14] implements the idea in [13] to run on-board a quadrotor. This implementation is made highly optimized for a edge-device to run full time.

There are several variations of Kalman filters to approximate the non-linearity in the system dynamics. [8] implements an unscented Kalman filter to estimate the orientation of a non-linear system. [15] and [7] implement an extended Kalman filter and error state Kalman filter respectively. [7] implementation is VIO based while that of [8] is only inertial odometry based filtering system. [16] describes a complimentary filter approach to solving the filtering problem for non-linear systems. The idea of using multiple filters to give a more robust estimate is introduced in [6].

E. Approach

1) The Filter Variants Implemented to Incorporate in the Ensemble: In this project we implement and ensemble the UKF [8], ESKF [7] and Complementary Filter (CF) [16]. We focus on orientation estimation since it is often the hardest part of the state to estimate. The datasets used for all analyses in this project come from the EuRoC MAV dataset [17]. The EuRoC MAV dataset has 5 datasets of varied complexity levels of drone flight. 2 datasets have a simple environment with less aggressive maneuvers labeled "easy", 1 dataset labeled "medium" and 2 datasets labeled "hard".

2) Training the Perceptron and Neural Network: The datasets are split 60-40 into training and tests sets. Machine Hall (MH) 01 (easy), MH-03 (medium) and MH-05 (hard) were used for training the ensemble network and MH-02 (easy) and MH-04 (hard) were used for testing.

The filters assume that the origin of the world frame is the initial pose of the drone at t_0, which is different from the Vicon (ground truth) world frame. In order to correctly compare against the Vicon system, a corrective translation and rotation was applied to all of the states of the filters based on the Vicon initial pose. The corrected estimates from the 3 filters are saved to later incorporate into the ensemble for training.

The orientation estimates from the 3 ensemble filters were stacked together along with the ground truth after aligning the timestamps of the filter outputs. Aligning the filter outputs is critical since the ESKF uses vision measurements and IMU propogation while the CF and UKF rely on the IMU only. Timestamp matching is performed by subtracting one timestamp of the filter from all the timestamps in the ground truth readings and then finding the argmin of the absolute difference to find the index of the reading to consider. We end up with 3 datasets of shape $(N \times 4)$, the 4th column being the ground truth labels, which will be used to train 3 perceptron networks and 3 neural networks, one each for roll, pitch and yaw. Each perceptron network takes 3 inputs at each timestep and produces 1 linear estimate. Each neural network takes 3 inputs at each timestamp and are passed through a hidden layer of dimension 5 with a ReLU activation function, which are then fully connected into a single estimate. The predictions are compared with the ground truth estimates using mean-squared-error (MSE) loss or custom loss function, which is then used for backpropagation and updating the weights. Since orientation estimates, namely roll, pitch, and yaw angles, are nonlinear, the choice of loss functions evaluate a linear distance loss (MSE) and an nonlinear function that considers the double-cover with angle wrap around. In particular, the loss between angles 359° and 1° should reflect an absolute distance of 2° and not 358°. The custom loss function takes the following equation:

$$loss = |((\theta_{estimate} - \theta_{truth}) + 180) \bmod (360) - 180| \quad (8)$$

The optimizer we have used is the Adam optimizer which takes in arguments for learning rate and a regularizing weight-decay parameter.

3) Testing and Validation of the Ensemble: The training dataset is split into training and validation datasets with a 80-20 split. Mini-batches from the training dataset are used to the train the network. Once the network is trained, it is verified with the verification set and saved offline for testing. The saved models are then tested on the datasets MH-02 and MH-04. The performance of the model is evaluated by computing MSE loss and the loss described in Equation 8 between the predicted states and the ground truth values. The states obtained from each filter, the ensemble state estimate, the simple average, and the ground truth were then be plotted together for evaluation.

F. Experimental Results

The following figures illustrate the results for the dense neural network ensemble with MSE loss, where ESKF is the green line, UKF is the pink line, CF is the red line, the ground truth is the black line, and the network output is the blue line.

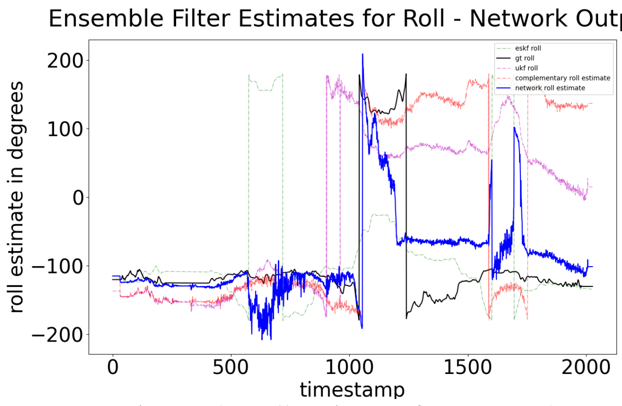


Figure 18: Roll Estimates for Dataset 4

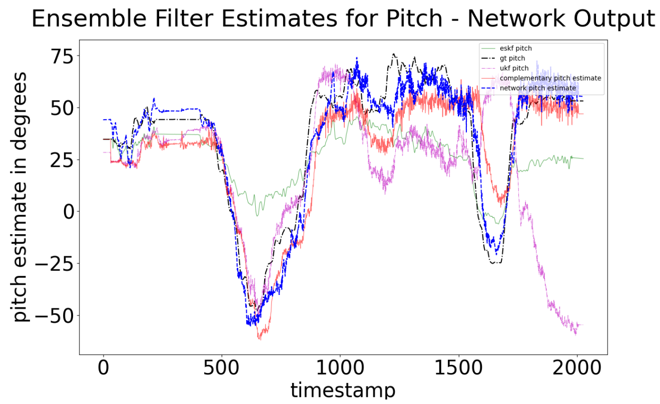


Figure 19: Pitch Estimates for Dataset 4

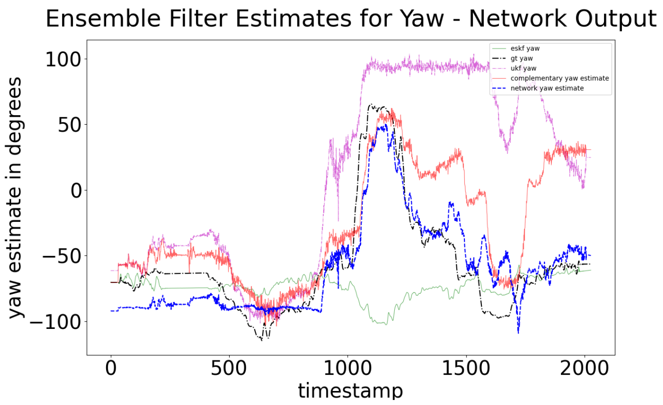


Figure 20: Yaw Estimates for Dataset 4

The following figures illustrate the results for simple average, dense NN with MSE loss and ground truth, where simple aevarge is the red line, dense NN is the blue line and the ground truth is the black line.

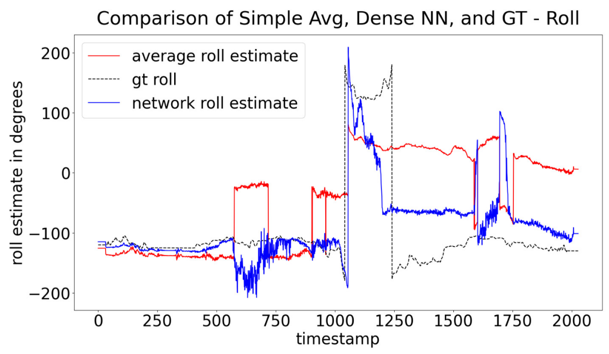


Figure 21: Simple average, Dense NN and GT comparision for Dataset 4 for Roll

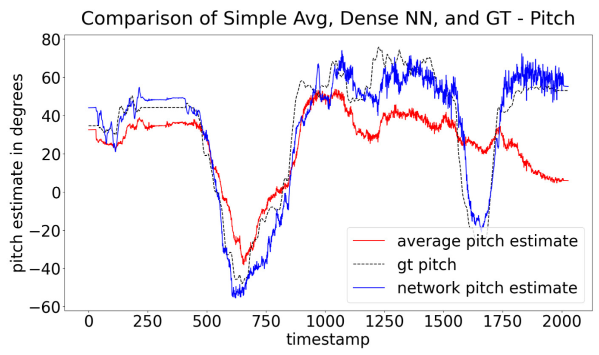


Figure 22: Simple average, Dense NN and GT comparision for Dataset 4 for Pitch

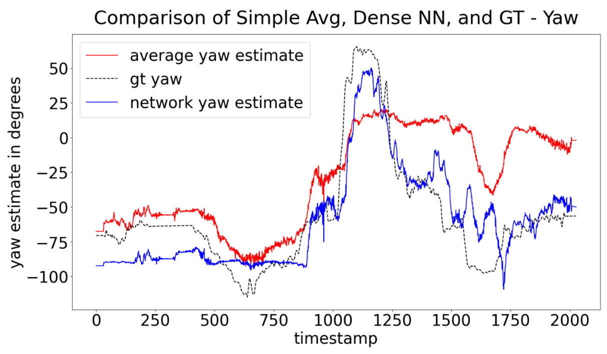


Figure 23: Simple average, Dense NN and GT comparision for Dataset 4 for Yaw

G. Discussion

Three methods were employed and tested to get the ensemble estimate from the three filters. The methods employed were simple average, perceptron network and a dense neural network. Simple average worked well in scenarios where one filter would be under-estimating the states while the other filters would be over-estimating the states. Cases when all the filters are either under-estimating or over-estimating, the simple average would not perform well. Therefore, simple averaging was not good for generalization, but roughly followed the correct trend.

This result motivated us to build a perceptron network, one each for roll, pitch and yaw. A linear activation function was used in the perceptron network in order to get a weighted average along with an Adam optimizer. The perceptron was able to generalize better compared to the simple average and handle the situation better when all the filters are either under-estimating or over-estimating. The only place where it would fail to work is when the quaternion flips, which is the same as the aforementioned double-cover of angles (angle wrap-around). From the instance the network is faced with an angle wrap-around issue, the error cascades to all the estimates after that instant and results in a poor ensemble estimation of states.

In order to deal with the set-backs experienced with the simple average and the perceptron, a few different approaches were tried to deal with angle wrap-around. One of the approaches was to try a different loss function. The loss functions that we compared were the MSE loss function and the custom loss function in Equation 8. The purpose of the custom loss function has been discussed in section 4.2. Upon evaluation, it was observed that the MSE loss performed better than the custom loss function since the angle wrap-around has not been dealt with in the ground truth data and hence, it was not required of the loss function to deal with this. The network is required to learn to fit a line as closely as possible to the ground truth and hence the MSE loss function performed better at this task.

Another solution to deal with the setbacks was to create a dense neural network. The architecture of the dense network has also been discussed in section 4.2. Since this was a complex network compared to a simple perceptron, it was able to capture the nonlinearity of anglular data and therefore generalize better to then predict estimates with a lower margin of error. It was still unable to handle the angle wrap-around perfectly, but even after the angle wrap-around instance, the error would not cascade to the future estimates and hence would be able to predict estimates relatively well for the future timesteps. For datasets that experienced no angle wrap around, the performance of the dense neural network was extremely good as seen in Figure 19 and Tables I II.

In Tables I II, it can be seen that the perceptron and dense neural networks generalized best for roll and pitch. However, yaw experienced the most instances of angle wrap-around and therefore generalized poorly. On the other hand, the custom loss function designed to handle angle wrap-around generalized best for yaw, but struggled with roll and pitch. Using the MSE loss function for roll and pitch and the custom loss function for yaw is expected to yield the best results.

Since the main challenge for network generalization was angle wrap-around, it can be expected that modifying each filtering algorithm to preprocess the double cover (i.e. unwrap the

RMSE between filter estimates and ground truth for roll, pitch and yaw - MH-02 DATASET			
Filter Name	RMSE for roll	RMSE for pitch	RMSE for yaw
UKF	142.51	21.70	137.07
ESKF	140.04	25.11	97.78
CF	146.53	28.79	109.26
Ensemble Filter (Simple Average)	128.14	13.75	98.17
Ensemble Filter (Perceptron - MSE Loss)	114.75	18.34	95.43
Ensemble Filter (Dense - MSE Loss)	113.59	11.22	130.89
Ensemble Filter (Dense - Custom Loss)	136.02	11.23	83.46

Table I: Summary of results - MH-02

RMSE between filter estimates and ground truth for roll, pitch and yaw - MH-04 DATASET			
Filter Name	RMSE for roll	RMSE for pitch	RMSE for yaw
UKF	144.98	42.37	87.84
ESKF	115.31	23.46	49.68
CF	157.72	17.68	43.34
Ensemble Filter (Simple Average)	108.25	23.67	42.27
Ensemble Filter (Perceptron - MSE Loss)	86.78	16.01	35.42
Ensemble Filter (Dense - MSE Loss)	64.88	10.18	20.75
Ensemble Filter (Dense - Custom Loss)	98.81	11.70	41.93

Table II: Summary of results - MH-04

angles) would yeild optimal results.

VI. Mini Minecraft
Fall 2022

A custom mini minecraft implementation from scratch using OpenGL, GLSL, C++, and Qt coming soon! This project will include player physics, procedural terrain generation, chunking and optimized world rendering, NPCs, camera transformations, path planning, and more! This was the most in depth project I have ever tackled, and everything you see in the video below was created entirely from scratch. Huge shoutout to my amazing teammates Evan Grant and Benedict Florence!

There are many implementation and role breakdown details in the GitHub README.md file found in the link below.

A. Video Link

VIDEO LINK

B. GitHub Link

GITHUB LINK

VII. Energy Conscious MPC in Fluid Flows
Fall 2022

A. Abstract

This project aims to develop methodology for controlling an energy conscious shape-shifting marine robot utilizing an MPC controller to capitalize on fluid flow dynamics to follow a desired trajectory. A linear MPC controller was developed around a fluid particle dynamics equation formulated by Maxey-Riley [18] by linearizing the dynamics and creating an optimization program, and controller performance was evaluated in a simulated vortex flow field. With control comprising robot density and radius, the robot was able to successfully reach a desired trajectory at a fixed radius from the vortex center, however the speed of convergence and stability of the controller were underwhelming. Tuning the MPC controller was difficult because a linear controller was built on top of nonlinear dynamics, and constructing a nonlinear solution is expected to yield better results in the future.

B. Introduction

A traditional way to design marine robots is to use hydrodynamics to create torpedo-shaped, high propulsion vessels (Poner cita). The most considerable trade-off of this kind of robot is its high energy consumption. There is an excellent research opportunity in the development of new energy-conscious vessels. The most standard way to design any marine robot is to use a Newtonian or Hamiltonian approach; these methods are presented by Thomasson, who gives an excellent summary and insight on using Classical Mechanics to calculate position, velocity, and acceleration [19]. The authors recapitulate three basic main motion models for a vehicle in a flow field. First, the kinematic particle model, described by eq. 9, where the vehicle dynamics are ignored, there is the consideration of a massless particle that moves and follows a given flow. The second model is the dynamic particle, described by eq. 10, where the vehicle's mass and the forces acting on it are considered but attitude dynamics are ingored - including the moment effects due to any present flow field. The third and more complete model is the Dynamic body, described by eq. 11, where mass, rotation, moments, and forces are considered. Going further with each model increases the difficulty of calculating dynamics for the vehicles but describes in a more significant way the object dynamics in a given flow. These methods typically use the Newtonian and Lagrangian methods to compute position, velocity, and acceleration equations.

Kinematic particle

$$\dot{X} = v_f(X,t) + v_{ctrl} \tag{9}$$

Dynamic particle

$$\begin{cases}\dot{X} = v_f(X,t) + v_r \\ m\dot{v}_r = f_o(X, v_r, v_f, t) + f_{ctrl}\end{cases} \tag{10}$$

Dynamic body

$$\begin{cases}\dot{X} = R(v_f(X,t) + v_r \\ \dot{R} = R\hat{\omega} \\ M\dot{v}_r = f_o(X, R, v_r, v_f(X,T), \omega) + f_{ctrl} \\ M\dot{\omega} = m_o(X, R, v_r, v_f(X,T), \omega) + m_{ctrl}\end{cases} \tag{11}$$

Using any of these approaches will give a suitable approximation of reality. There are two main trade-offs to using these kinds of models. The first one is the complexity of solving this kind of system, and the second is the difficulty of including effectively time-dependent flow fields. If the flow field effects were considered, they need to be computed and analyzed as external forces and moments. Another exciting and robust approach is to use fluid mechanics to analyze the behavior of particles on given flows. One of the most accepted approaches is the Maxey-Riley (MR) model, which describes the motion of a small rigid sphere in a nonuniform flow. This technique presents an excellent advantage by considering the external influences over a particle [18]. This model is presented in eq. 12.

$$\begin{aligned} m_p\frac{dV_i}{dt} = (m_p - m_f)g_i + m_f\frac{Du_i}{Dt} \\ -\frac{1}{2}m_f\frac{d}{dt}\{V_i(t) - u_i[Y(t),t] - \frac{1}{10}a^2\nabla^2 u_i|_{Y(t)}\} \\ -6\pi a\mu\{V_i(t) - u_i[Y(t),t] - \frac{1}{6}a^2\nabla^2 u_i|_{Y(t)}\} \\ -6\pi a^2\mu\int_0^t d\tau(\frac{\frac{d}{d\tau}\{V_i(\tau) - u_i[Y(\tau),\tau] - \frac{1}{6}a^2\nabla^2 u_i|_{Y(\tau)}\}}{[\pi\nu(t-\tau)]^2}) \end{aligned} \tag{12}$$

In this equation, a is the sphere's radius, m_p is the sphere's mass, and m_f is the mass of the fluid. The fluid and particle have the same volume as the sphere. $Y(t)$ is the instantaneous position of the particle, $V_i(t)$ is the instantaneous velocity of the particle $u_i(x,t)$ is the undisturbed flow field $u_i(x,t)$ depends on the position x and time, μ and ν are the dynamic and kinematic viscosity, respectively. It is important to include the recommendation from Riley. Because, in the primary form, the MR equation has two different time derivatives, the first one is a derivative that follows the moving sphere, eq. 13 and the second one is a derivative that follows the moving fluid, eq. 15.

$$\frac{d}{dt}u_i\left[Y(t),t\right] = \left(\frac{\partial u_i}{\partial t} + V_i\nabla u\right) \tag{13}$$

$$\left.\frac{Du_i}{Dt}\right|_{Y(t)} = \left(\frac{\partial u_i}{\partial t} + u_i\nabla u\right) \tag{14}$$

Riley recommends, the first term has to be substituted by:

$$m_f\left.\frac{Du_i}{Dt}\right|_{Y(t)} \tag{15}$$

The first term of MR equation is the gravity force F_g and represents the buoyancy of the particle relative to the fluid.

$$F_g = (m_p - m_f)g_i \tag{16}$$

The second term is the fluid acceleration force F_{fa}; This is the force given by the acceleration in the center of the particle. It accounts for the pressure gradient of the undisturbed flow.

$$F_{fa} = m_f\left.\frac{Du_i}{Dt}\right|_{Y(t)} \tag{17}$$

The third term is the added mass force F_{am}; this is the inertia added to the system. It is a result of the movement of the particle and the fact that this particle has to haul the same amount of fluid to occupy a space.

$$F_{am} = -\frac{1}{2}m_f\frac{d}{dt}\left\{V_i(t) - u_i\left[Y(t),t\right] - \frac{1}{10}a^2\nabla^2\left.u_i\right|_{Y(t)}\right\}$$

The fourth element is the Stokes drag force F_{sd}; this is the drag force and occurs on small spherical particles with a small Reynolds number in a viscous fluid.

$$F_{sd} = -6\pi a\mu\left\{V_i(t) - u_i\left[Y(t),t\right] - \frac{1}{6}a^2\nabla^2\left.u_i\right|_{Y(t)}\right\} \tag{18}$$

The last element is the Basset history term F_{bh}; this is a memory effect and describes the cumulative effect of the diffusion of the vorticity from the particle along its entire path.

$$-6\pi a^2\mu\int_0^t d\tau\left(\frac{d/d\tau\left\{V_i\tau - u_i\left[Y(\tau),\tau\right] - 1/6a^2\nabla^2\left.u_i\right|_{Y(\tau)}\right\}}{\left[\pi\upsilon\left(t-\tau\right)\right]^2}\right) \tag{19}$$

A required piece of information is the inclusion/exclusion of the Faxen correction, which is included in each term and is related directly to the laplacian operator $\nabla^2\left.u_i\right|_{Y(t)}$.

Given its difficulty, studies are done on when to use the terms presented in the Maxey-Riley equations. Some of them are [20], [21], [22]. The use and exactitude of the Maxey-Riley equations have also been shown [23]. For the case of this paper, we follow the assumptions from [21] that: first, the gravity term is ignored because we are going to analyze a 2D system where gravity is not present second, the Basset history term can be omitted when the viscosity is small, third, the Faxen correction is essential only in regions with significant vorticity, and for now Basset history and Faxen correction will be neglected. Some recent studies showed that Basset history and Faxen correction could contribute more significantly than believed before [24], [22].

After getting these assumptions, the Maxey-Riley equation can be presented as 20; this representation is advantageous to understanding the behavior of a particle depending on the fluid properties (mass and viscosity) and the particle properties (mass and radius).

$$m_p \frac{dV_i}{dt} = m_f \left. \frac{Du_i}{Dt} \right|_{Y(t)} - \frac{1}{2} m_f \frac{d}{dt} \left\{ V_i(t) - u_i \left[Y(t), t \right] \right\} \\ -6\pi a \mu \left\{ V_i(t) - u_i \left[Y(t), t \right] \right\} \tag{20}$$

This is the mass formulation of the Maxey-Riley equation given in dimensional form. This equation can be used directly; for a more neutral approach, the dimensionless formulation of the Maxey-Riley equation is necessary. This paper uses the density formulation to build the dimensionless representation. The first step to get this representation is to divide the equation by $4/3\pi a^3$, which is the volume of a sphere of radius a, then reorganize the equation to get all the terms of the particle acceleration on one side, giving the following representation.

$$\rho_p \frac{dV_i}{dt} + \rho_f \frac{1}{2} \frac{dV_i}{dt} = \rho_f \left. \frac{Du_i}{Dt} \right|_{Y(t)} + \frac{1}{2} \rho_f \frac{d}{dt} \left\{ u_i \left[Y(t), t \right] \right\} \\ - \frac{9\nu\rho_f}{2a^2} \left\{ V_i(t) - u_i \left[Y(t), t \right] \right\} \tag{21}$$

Then dividing the equation by ρ_f and minimizing similar terms, we get the next representation.

$$\left(\frac{2\rho_p + \rho_f}{2\rho_f} \right) \frac{dV_i}{dt} = \frac{3}{2} \left. \frac{Du_i}{Dt} \right|_{Y(t)} - \frac{9\nu}{2a^2} \left\{ V_i(t) - u_i \left[Y(t), t \right] \right\} \tag{22}$$

Here it is clear that the $\frac{2\rho_f}{2\rho_p + \rho_f}$ is a dimensionless constant that is going to be represented as R, and $\frac{9\nu}{2a^2}$ is very close to the Stokes number that is $St = \frac{2a^2 U_o}{9\nu R L_o}$ where L_o is the characteristic length and U_o is the characteristic flow. The following equation is found when representing the complete Maxey-Riley equation in its dimensionless form.

$$\dot{v} = \frac{3}{2} R \left. \frac{Du_i}{Dt} \right|_{Y(t)} - \frac{1}{St} \left\{ V_i(t) - u_i \left[Y(t), t \right] \right\} \tag{23}$$

This is the dimensionless Maxey-Riley equation. Here it is evident that $\frac{2}{3}R$ is a critical factor when the particle has the same density as the fluid $R = \frac{2}{3}$, when $R < \frac{2}{3}$ particles are heavier than fluid (Aerosol), and $R > \frac{2}{3}$ the particle is lighter than the fluid (Bubbles). Another important fact is that the second term depends linearly on the Stokes number. Here it is necessary to notice that when $St \approx 0$, the particle $\dot{v}$ tends to follow the flow direction. In this paper we use this equation to model a shape-shifting robot where the radius and the density are controlled to navigate a given flow field.

C. Methodology

We use the eq. (23) as the Flowbot's dynamics and this equation was linearized in order to implement a MPC controller. First, (24) is the same equation, with a nomenclature that is closest to the one we use in class, different from the traditional fluids convention.

$$\dot{v}_p = \frac{3}{2} \left(\frac{2\rho_f}{2u_1 + \rho_f} \right) \left(\frac{\partial v_f}{\partial t} + v_f \nabla v_f \right) - \left(\frac{9\nu\rho_f}{u_2^2 (2u_1 + \rho_f)} \right) (v_p - v_f) \tag{24}$$

Where the subscript p denotes the robot/particle and the subscript f denotes the fluid in which the robot is in. u_1, u_2 are the robot density and robot radius respectively. ν is again the kinematic viscosity of the water, ρ is density, and v is velocity. The state of the robot was assumed to be $\mathrm{x} = \begin{bmatrix} x & y & \dot{x} & \dot{y} \end{bmatrix}^T$. These dynamics were linearized around the desired state and control to achieve the form $\dot{\mathrm{x}}_e = A\mathrm{x}_e + B\mathrm{u}_e$. Where $\mathrm{x}_e = \mathrm{x} - \mathrm{x}_d$ defines error coordinates.

$$A = \frac{\partial f}{\partial \mathrm{x}} |_{x_d, u_d} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & -\left(\frac{9\nu\rho_f}{u_{2d}^2 (2u_{1d} + \rho_f)} \right) & 0 \\ 0 & 0 & 0 & -\left(\frac{9\nu\rho_f}{u_{2d}^2 (2u_{1d} + \rho_f)} \right) \end{bmatrix}$$

$$B = \frac{\partial f}{\partial \mathrm{u}} |_{x_d, u_d} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ \mathrm{I}_x & \mathrm{II}_x \\ \mathrm{I}_y & \mathrm{II}_y \end{bmatrix}$$

$$\mathrm{I} = \frac{\partial \ddot{\mathrm{x}}}{\partial u_{1d}} = -\frac{6\rho_f (\dot{v}_f + v_f \nabla v_f)}{(2u_{1d} + \rho_f)^2} + \frac{18\nu\rho_f (\dot{x}_{(x,y)d} - v_f)}{u_{2d}^2 (2u_{1d} + \rho_f)^2}$$

$$\mathrm{II} = \frac{\partial \ddot{\mathrm{x}}}{\partial u_{2d}} = \frac{18\nu\rho_f (\dot{\mathrm{x}}_{(x,y)d} - v_f)}{(2u_{1d} + \rho_f)} u_{2d}^{-3}$$

There are several basic fluid flows which are commonly simulated, including vortex, vortex with sink, gyre, and double gyre cases. For simplicity in this work, complete knowledge of the structure is assumed and a circular vortex is used to evaluate controller performance. The desired state $\mathrm{x_d}$ may be defined as any arbitrary trajectory in the 2D fluid field, and for now the positional components of $\mathrm{x_d}$ are computed to follow a constant radius about the vortex center, with the velocity components aligning with the fluid flow.

Linearizing about a desired trajectory using error coordinates requires providing desired input u_d. Because a more passive robot is desirable in this application, u_d is set to the control input at the previous time step.

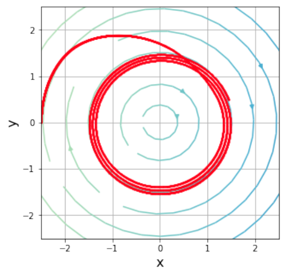
Once the linearization was complete, the MPC was set up such that there was a time horizon of $N = 5$ steps and $dt = 0.1$ seconds. The horizon is short because of the high non-lineairty of the dynamics.The LQR cost formulation for the finite horizon was cost $= \frac{1}{2}\mathrm{x}_e^T Q \mathrm{x}_e + \frac{1}{2}\mathrm{u}_e^T R \mathrm{u}_e$ The optimization problem is given by

$$\begin{aligned} \min_{x,u} \quad & \text{quadratic cost,} \\ \text{s.t.} \quad & \text{initial state constraint,} \\ & \text{input saturation constraint,} \\ & \text{dynamics constraint.} \end{aligned} \tag{25}$$

D. Results

Figures (24), (25), (26), (27) depict the 2D trajectory of the robot in the fluid flow field, a 1D representation of the radius of the robot in the fluid flow, and plots of the control inputs over time.

As shown in Figures (24) and (25), the linear MPC controller was able to achieve decent tracking of the desired radius when moving outward and moving inward in the vortex fluid flow. In both cases, it takes the robot a long time to converge ($\approx$ 250 sec) to the desired radius because of the lack of powerful actuators. The blue lines represent a moving average of the radius to clearly show that the robot was converging towards the intended vortex radius. The red lines represent the robot's radius without a moving average filter, which can be seen having some oscillation.


(a) Particle moving inward

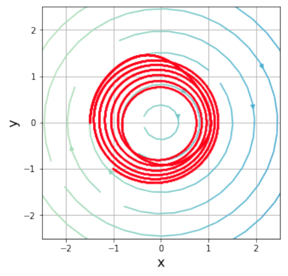
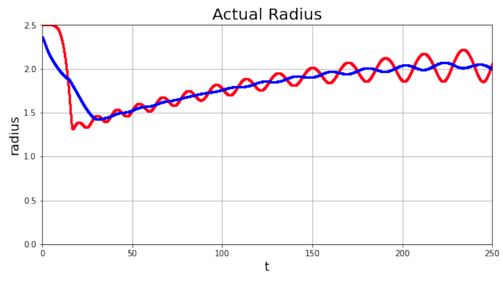

(b) Particle moving outward

Figure 24: 2D representation of particle vortex radius.


(a) Particle moving inward

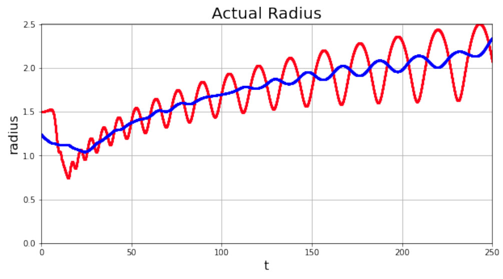
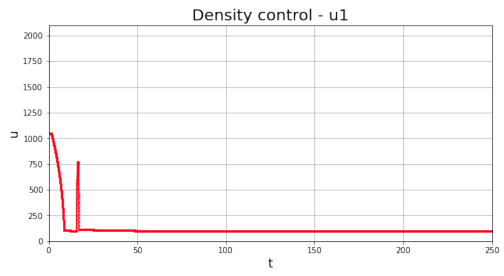

(b) Particle moving outward

Figure 25: 1D representation of particle vortex radius.

Figures (26) and (27) show the control inputs over time. It can be seen that the robot preferred one mode of control (radius) over the other (density) by first decreasing density completely until the controller learned to used a nearly bang-bang radius controller.


(a) Particle moving inward

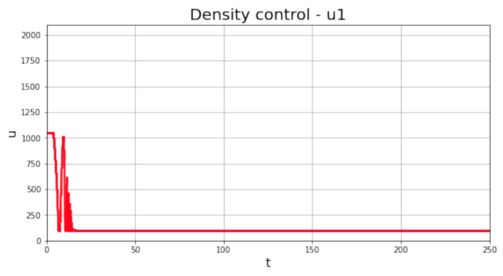
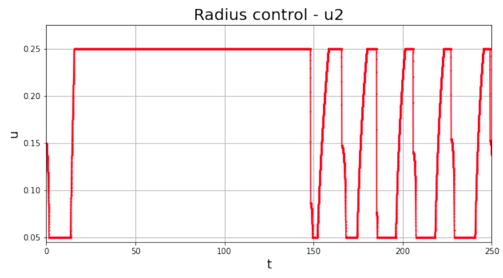

(b) Particle moving outward

Figure 26: Density control of the robot over time


(a) Particle moving inward

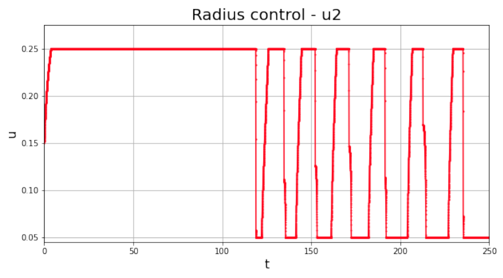

(b) Particle moving outward

Figure 27: Radius control of the robot over time

In both inward and outward control, the robot was incredibly slow as expected.

E. Discussion

We suspect that linearizing the highly nonlinear dynamics of this system inhibited the effectiveness of the controller and resulted in a system which is inherently not control-affine. This approximation could explain why only a very short MPC time horizon provided reasonable control output. For example, as evident in Figure (26), the controller frequently drove particle density to the minimum. With a shorter time horizon, the controller began employing density control but often in a destabilizing manner.

One improvement to methodology considered for future work is nonlinear MPC using the full dynamics and more robust solver (i.e. SNOPT). Additionally, this could avoid having to set a relatively arbitrary u_d which is nontrivial to generate if the previous timestep control is not used.

Another option could be an augmented state space - adding u to the state vector and instead making the control input $\dot{u}$ such that the linearized dynamics become control-affine. This would also be conducive for easily imposing an energy-saving penalty on a change in control input over time.

The oscillation in Figure (25) which arises as the particle approaches the desired trajectory comes from eccentric flow around the vortex center which is biased toward the initial position of the particle. During tuning, this behavior worsened when applying large positional cost in Q, thereby demanding higher control effort, and when using a longer time horizon. Other approaches like those mentioned above might allow us to extend our time horizon with more accurate prediction, and we expect that this oscillation would diminish as the cost incorporates the lower frequency error of the oscillation.

F. Conclusion

A linear MPC controller to control a shape-shifting robot in a highly nonlinear flow field was only marginally successful after extensive tuning. Although the robot did converge towards the desired goal, it is recognized that the approach taken is likely not optimal because of the difficulties that came with linearizing the dynamics (dynamics were not control affine). There are many assumptions that had to be made that built up over time, making the process for getting a working controller difficult. Picking u_d for an energy efficient trajectory made sense in this application. However, it is still ultimately arbitrary because choosing $u_d = u_{d_{t-1}}$ was a conscious design choice that was not proven effective with mathematical rigor (which is very difficult to do). Additionally, choosing the time horizon, step size, initial state, and the Q/R matrices had significant impact on the controller's performance and were highly sensitive parameters to tune. Because of the many challenges with a linear MPC controller, it is concluded that moving towards a nonlinear MPC controller is a better strategy in the future to eliminate the simplifying assumptions that deviate the from the true dynamics of the system.

VIII. Music Generation Using Deep Learning - Variational Autoencoders Fall 2022

A. Abstract

This project developed an architecture for music generation conditioned on music consisting of Variational Autoencoders, a Convolutional Neural Network, and a Multi-Layer Perceptron Network. Using song data from the Lakh Pianoroll Dataset [25] and the Million Songs Dataset [26], sequences across 5 instrument channels, namely piano, guitar, strings, bass, and drums, were generated from the output of the decoders of the VAEs. This work aimed to improve upong and verify the legitimacy of [27], which included many flaws. It was concluded that their work was illegitimate, however with an improved architecture conditioned on genre we were able to produce promising music sequences.

B. Introduction

Deep learning has transformed the fields of artificial generation in Computer Vision with deep fakes, NLP with article generation, and robotics with trajectory generation. But what about artistically generating audio signals using deep learning? Music generation would involve learning the music sequence and generating further the sequence of the fed music data. This problem is interesting because it allows us to explore the intersection of Deep Learning and the art of music, which is an integral part of our daily lives. Music can be reduced to a mathematical sequence of events which can thus be represented by nodes. What makes music generation difficult is the temporal nature of sounds to form a subjectively "good" melody and seamlessly blend different channels of tones (i.e., different instruments, genres, voices, etc.). Typically, state-of-the-art music generation architectures include RNNs, VAEs, GANs, and Transformers. In this project, we aim to improve the existing architectures of VAEs for music generation and to focus on the conditional generation of music based on genre.

C. Background

Our aim in this project is to leverage the power of novel neural network architectures that will help facilitate the generation of new music. We will be using the popular Lakh Piano Dataset [25] in addition to the Million Song Dataset [26]. Music provides different challenges in the pipeline of processing since it consists of different instruments that are dependent on each other, along with the added dependency of time. There is more structure to music than just this; a few of them are chords and melodies, which would result in each time step having many outputs. But audio data also assumes many properties that help model this problem in a similar manner to other areas of study. The sequential aspect can be modeled using an RNN, whereas the multiple channels of audio tracks can mimic the channels of an image - which can be modeled using CNNs. Much of recent work also uses deep generative models, which are upcoming and seem to do a good job of creating synthetic music that appears to be real.

We primarily use a VAE in combination with a few linear layers and activation functions like ReLU with dropout. The benefits of using VAE are manifold; it is essentially a regularized autoencoder that provides a latent space with beneficial properties in a probabilistic manner, and this enables a generative process.

D. Related Work

Our main architecture is an extension of the work done by Isaac Tham [27] on Generating Music Using Deep Learning. Their baseline method involved different approaches; from next-note prediction with RNNs, multi-instrument RNN, and later moving to CNNs which modeled 5 different CNNs (one for each input) that helped the generation of new music in an iterative process when given a starting multi-instrument sequence. These methods, however, displayed only very little variation in the generated music, where most generations seemed identical to each other. When they introduced the variation autoencoder method, they were able to map inputs into latent distributions, and this helped introduce variation to the generated music. While the work done previously in the larger context of literature models ranging from LSTMs to diffusion models, have been tried and tested. The problem we are trying to analyze and attempt to solve is conditional music generation, where the model conditions the output of the music to be generated on the genre of the track in consideration.

E. Approach

Music can be divided up into multiple channels, and the channels that were distinct and useful for this project include piano (which constitutes the melody), guitar, bass, strings, and drums. Using the Lakh Pianoroll Dataset [25] and the Million Songs Dataset [26], genres were matched to track identifiers such that each song had a corresponding genre for training. Features such as type and number instruments used and song length were used in the training process. The aim was to improve upon the work of [27] which had many flaws in their approach. For example, the test data in which they had generated music from was part of the training set, and thus it was likely that their generated music was not genuine. Additionally, it was assumed in the VAEs that the variance of the latent faactor was assumed to be a uniform single number for each instrument, which is a bold assumption. The VAEs were reformulated by maximizing the objective in (26).

$$\mathrm{ELBO}(u,v;x^i) = \mathop{\mathrm{E}}_{z\sim q_u(z|x^i)}[log(p_v(x^i|z)] - \mathrm{KL}(q_u(z|x^i)||p(z))$$

$$u^*, v^* = \operatorname*{argmax}_{u,v\in\mathbb{R}^p} \sum_{i=1}^{n} \mathrm{ELBO}(u,v;x^i) \tag{26}$$

The encoder parameters are represented with the variable u and the decoder parameters with variable v. The encoder outputs latent factor $z \in \mathbb{R}^k$ where $k \in \{16, 32, 64, 128\}$, where half of each space are neurons for the means and the other half are neurons for the standard deviation for an assumed Gaussian distribution. Using these latent factors z, a deep network that takes as input the latent factors z^t is used to predict the latent factors at the next timestep z^{t+1}. The network was chosen as Convolution Neural Network (CNN) for the piano channel, which was conditioned on a user specified genre. Because this neural network existed inside of the latent space, the Conditional CNN (CCNN) acted as if the VAE itself was conditioned, as expressed in (27).

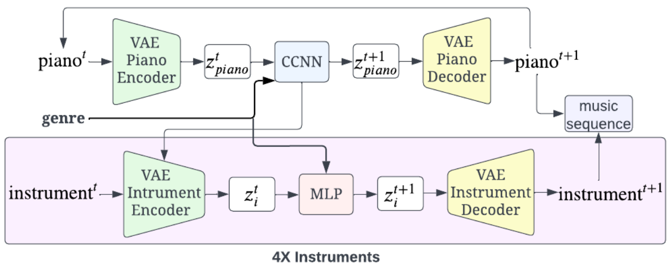


Figure 28: Conditional Music Generation Architecture

$$\begin{aligned}\mathrm{ELBO}(u,v;x^i,c) = &\mathop{\mathrm{E}}_{z\sim q_u(z|x^i,c)}[log(p_v(x^i|z,c)] \\ &- \mathrm{KL}(q_u(z|x^i,c)||p(z|c))\end{aligned} \tag{27}$$

This architecture of a CCNN inside of a VAE was constructed only for the piano channel since it was being used as the melody to the songs since every song in the Lakh Pianoroll Dataset [25] utilized piano, where the other channels were not always present. Similar to the piano channel, 4 VAEs were constructed for the bass, strings, guitar, and drums channels that were conditioned on genre and the z^{t+1}_{piano} to again maximize the conditioned objective in (26, 27). However, instead of a CCNN inside of the latent space, a Multi-Layer Perceptron (MLP) neural network was used to predict the latent factors z^{t+1} given inputs z^t for its respective instrument. For all channels, there was some slight noise injected into the latent factors z^t to encourage novel latent factors z^{t+1} such that unique music was generated. Finally, the decoder took the ltent factors z^{t+1} and appended to a music sequence.

In total, there are 5 VAEs, 1 CCNN, and 4 MLPs that required training. Out of the 15 possible genres that are part of the Million Songs Dataset [26], there were 4 genres that felt distinct enough to be clearly distinguishable: rock, pop, electronic, and RnB. A subset of songs that were of these 4 genres were used for training. Figure (28) clearly illustrates how all of the networks fit together to generate music sequences.

In order to ensure the gradient is computable for the expectation term in (27), the reparameterization trick was used in (28). To keep notation clean, $log(p_v(x^i|z,c))$ was written as $\phi(z)$.

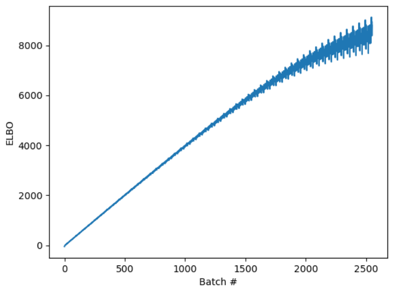
$$\begin{aligned}
&\nabla_u \mathop{\mathrm{E}}_{z\sim q_u(z|x^i,c)}[\phi(z)], \text{ where } \epsilon^j \sim N(0,I) \\
&= \nabla_u \mathop{E}_{\epsilon\sim N(0,I)}[\phi(\mu(x^i;u,c)+\sigma(x^i;u,c)\odot\epsilon)] \\
&= \sum_{j=1}^{N} \nabla_u \phi(\mu(x^i;u,c)+\sigma(x^i;u,c)\odot\epsilon)
\end{aligned} \tag{28}$$

The equation in (27) is now deterministic and thus backpropogation can be computed. The conditioning was accomplished by one-hot encoding the genres rock, pop, electronic, and RnB such that they were deterministic. Some songs exemplified more than 1 genre, and those songs were encoded with a multi-hot encoding, which was used to generate blended genres during generation.

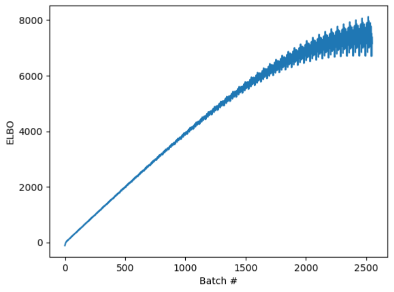
F. Experimentation

1) Data Preprocessing: For our experiments, we use the LPD5-cleansed dataset curated by the Music and AI Lab at the Research Center for IT Innovation, Academia Sinica [25]. This dataset consists of 5 track pianorolls (piano, drums, guitar, bass, and strings) for 21k songs. For the genre information, we used the Million Songs Dataset [26]. We matched the track ids for both datasets and found 11k common tracks with genre information available. We chose to use the top 4 genres with the highest count in the dataset, which were rock, pop, electronic and RnB for our project. Many songs had more than one label associated with them. We chose to represent such songs with a two-hot encoding. The others were represented using a one-hot encoding. We then split the tracks into its 5 instrument components and stacked each component for 100 randomly sampled songs. The tracks were then broken down into 32 length sequences, and each sequence was used as a single data point for each component.

2) VAE Training: We trained 5 VAEs, one for each instrument, using the processed dataset mentioned above. The genre tags were not used for training. The ELBO was computed for the 32 length sequences treated as a single unit. We experimented with different dimension sizes for the latent space(8,16,32,64). The rest of the experiments were conducted using 64 as the dimension of the latent space. In their experiments, [27] used assumed that the variance of all the independent gaussians is equal. We chose to use the more general case and set an output size of 128(2*64) for the encoder. A batch size of 128 gave the most stable convergence for the ELBO. We trained the VAEs for 50 epochs. The ELBO plots for a few instruments are shown in the figures below.



(a) ELBO Piano



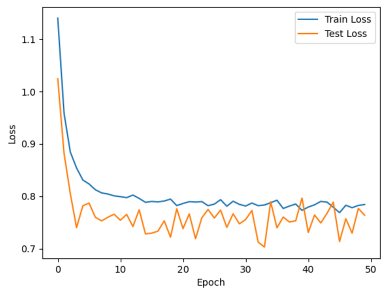
(b) ELBO Guitar

Figure 29: ELBO plots

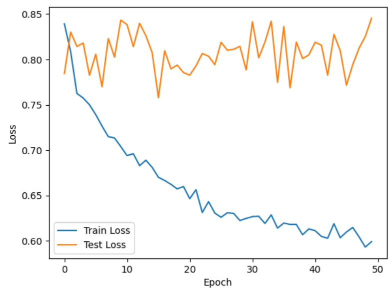
3) Neural Network Training: As described previously, we use a total of neural networks(one for each instrument). The piano and other instrument neural networks are trained in a slightly different manner. We first train the piano NN to take the sampled output from the piano VAE encoder for the current set of 32 notes as wekk as the genre encoding as input, and predict the sampled output from the piano VAE for the next set of 32 notes. In this way, we are teaching the piano NN to learn to generate future notes, but in the latent space. We trained the piano NN for 50 epochs using the Adam optimizer as well as SGD with Nesterov's momentem with a learning rate of 0.001 and momentum coefficient of 0.9. Adam resulted in a very high overfitting model as compared to SGD, and trying out a few regularization techniques like increasing dropout or adding weight decay did not help a lot. We chose to train both models for music generation. The overfit model with Adam would hopefully generate music similar to the existing ones, with variations coming from the VAE instead. The plots for the train and validation loss for both is shown in figure 3.

The NNs for other instruments took the genre encoding, VAE encoder sampled output for the current set of notes for that instrument, as well as the VAE encoder sampled output for the piano notes as input. In this way, they treated the piano melody as the base to generate their music. Similar to the piano NN, they were trained using SGD with Nesterov and Adam optimization for 30 epochs. The plots for the SGD train and validation loss are shown in figure 4.

4) Generating Music: In order to generate music using the the trained models, we first created a dataset of the first sequence (32 length) of all the tracks from the training data. We then sample 1 datapoint from it at random, on which the music is then generated. We also need a genre label(one-hot or multi-hot depending on whether we want to generate a song as a mixture of genres) which is chosen manually. This starting sequence is then broken down into its instrument components and passed through the instrument VAEs respectively. We then sample a random point from the latent distribution using the reparametrization trick and pass the sample along with the genre encoding to the piano NN. The other instrument NNs take the genre encoding, their own latent sample and the output of the piano NN as input. The outputs of all 5 NNs are then passed to the respective instrument VAEs. The outputs of all 5 VAEs are then stacked together to form the pianoroll, which is then converted to a midi file using the prettymidi and fluidsynth libraries. As we generate music in this manner, starting from a sampled song, the first few seconds of the generated track are the original song itself, after which the music generated by our model begins.

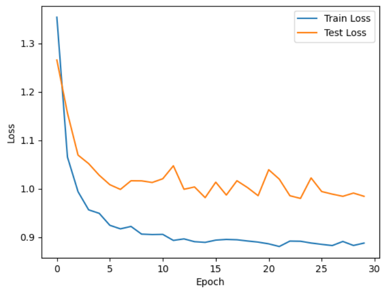


(a) SGD optimizer

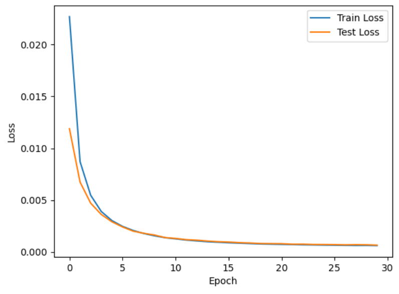


(b) Adam optimizer

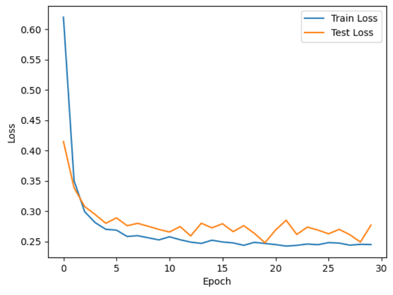
Figure 30: Piano CCNN



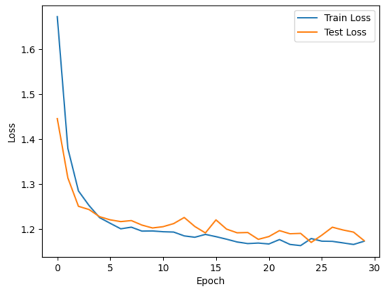
(a) Guitar Loss



(b) Bass Loss



(c) Drums Loss



(d) Strings Loss

Figure 31: SGD Losses for instrument NNs

G. Discussion

Table (**??**) has some links to results (click "'Link"' to listen) so that the reader can listen to the produced results.

Genre	Optimizer	Link
Electronic-Pop	SGD	Link
Electronic-Pop	Adam	Link
Rock-Pop	SGD	Link
Rock-Pop	Adam	Link
Rock	SGD	Link
Electronic	SGD	Link
Pop	SGD	Link
Pop	Adam	Link
RnB	SGD	Link

Table III: Generated music with genre conditioning and genre blending.

The quality of generation was very sensitive to initialization because the optimization routine was likely not perfectly tuned, and thus most generated songs were not dense with notes. Since there were many networks that required training, it required that all were training well in order to get a subjectively quality song, which is a non-trivial task. About $25-50\%$ of generated songs were subjectively a good listen, while the rest were either too slow (notes only every few seconds) or sparse. Additionally, generating subjectively good music where the genres were blended together was more difficult because there were much less samples in the training data. The choice of optimizer *subjectively* made similar sounding music, however the figures illustrated in (VIII-F) indicate that SGD is better on the test data, and thus SGD was the preferred optimizer.

We still chose to experiment on music generation with the overfit model with the Adam optimizer, and saw that it generates dense notes only for a a single or 2 instruments depending on the genre. For example, the song generated using electronic-pop as the genre and Adam optimizer consists of only a drum track, but is quite an accurate representation of the drums in electronic-pop songs. Another example of this can be seen in the song generated using pop as the label and Adam as the optimizer, where it consists of just a piano track that sounds accurate.

There was a very high class imbalance in the genre labels of the tracks. Multi-genre songs sounded better than single genre songs as the dataset had many songs with more than 1 genre assigned to it. RnB had the fewest training examples, and that can be seen reflected in the quality of the generated music.

There are many possible culprits as to why the generated music was not as rich in note density for all instruments as seen in [27], most notably being that their test set was seen within the training set so they effectively had knowledge of the next notes. It is also possible that conditioning the VAE's instead of the internal CCNN and MLP within the latent space could generate better conditioned latent factors, and thus provide a better next-note generation across all instruments. Recent works such as ([28], [29], [30]), although having different architectures all together, often incorporate more features such as note density, loudness, energy, tempo, acousticness, liveness, valence, emotion, etc. which have demonstrated better success in generation. In the future, trying different architectures such as transformers, different ways to condition (such as conditioning the encoder/decoder of the VAE mentioned previously), and incorporating more features would like to be explored to generate the best possible results. Additionally, there is not much work in literature that incorporates vocal tone as a channel along with instruments, which can dramatically change the sound of a song. Using that as a feature may be able to yield a more rich sound overall.

IX. Maya Lite
Fall 2022

A custom maya lite clone implementation from scratch using OpenGL, GLSL, C++, and QtThis project includes subdivision techniques (Catmull-Clark) and skinning. Visuals coming soon!

A. *Video Link*

Coming soon!

B. *GitHub Link*

GITHUB LINK

X. GLSL Shaders
Fall 2022

Surface and postprocessing shaders created using OpenGL, GLSL, C++, and Qt on a model of Mario (Wahoo)!

A. *Video Link*

VIDEO LINK

B. *GitHub Link*

GITHUB LINK

XI. Reinforcement Learning for Quadrotors: CIS 519 Final Project Spring 2022

A. *Full Report*

The report can be found fully written here (not rewritten here to preserve formatting): REPORT LINK

B. *GitHub Link*

GITHUB LINK

XII. 3D Scene Flow
Fall 2021

A. *Abstract*

This project focuses on the applications of feature correspondence, homography estimation, human pose detection and mesh transformations. Given a video of an individual moving, a rigid 3D mesh is to be warped to the surface of a specified body part which moves with the body throughout the video. This movement takes into account the 2D movement across the image plane as well as the rotation about any axis that occurs in the video. For example, if the desired goal is to give a human Iron Man's helmet, then the head must be tracked in order to warp the helmet onto the head with the correct pose. The 3D pose of the object must be maintained within the 2D video, hence 3D scene flow. Traditional methods tend to struggle with accommodating changes to the 3D orientation

of the subject, something we plan to explore in this work.
Collaborators: Aadit Patel, Rithwik Udayagiri, Ankit Billa, Evan Grant, Daniel Stekol

B. Goal & Objective

This project is split into 2 main goals/objectives -

1) Successfully track the 3D movement of the subject in image plane. This will involve human pose detection and feature point extraction from the subject, both of which are being taken care of using Google's MediaPipe framework [31].
2) Stick the external 3D object mesh onto a specified area of interest on the subject. We will compute the transformation matrix of the subjects 3D movements between 2 consecutive frames and using it to perform a rigid body transformation on the external object.

C. Related Work

- **Human Pose Estimation:** The authors of [32] released and demonstrated a novel method for real-time 2D multi-person pose estimation - and open-source library called OpenPose. This method takes a 2D RGB image input and outputs the 2D locations of anatomical keypoints of human figures in the image. This is done using a multi-stage CNN to first predict a set of confidence maps for body part locations, then generate part affinity fields which indicate the degree of association between body parts which comprise a single pose.
- **Rigid 3D Scene Flow:** The method presented by [33] involves a deep architecture capable of reasoning at an object level to describe a dynamic 3D environment using generalized optical flow. This strategy separates background (static) elements from dynamic foreground rigid body agents, and relaxes the supervision requirements for dense scene flow as a result of the object-level abstraction.
- **MediaPipe:** 3D human pose estimation with feature tracking for face and hands has been implemented in [31].
- **Optical Expansion:** The authors of [34] present a technique for using dense optical expansion - a cue of depth given by expansion of an image feature in the 2D image frame - for upgrading 2D optical flow to 3D scene flow.
- **3D SIFT Descriptor:** The authors of paper [35] propose a 3D SIFT Descriptor that can be used for action detection
- **Fourier Features to learn High Frequency functions**: [36] talks about using a Multi Layer Perceptron to learn high frequency parts after passing features through fourier functions.

D. Proposed Method

Our proposed approach can be divided into 5 sub-tasks: Human Pose Detection for Feature point extraction, load and render .OBJ file (3D object), scale and align .OBJ file to match first frame, calculate rotation and translation between frames, transform 3D object, and render each frame with 3D object superimposed.

E. Experimentation and Evaluation

1) OpenPose: The OpenPose library [32] was the initial candidate for the first step of the project: human pose detection. The openpose library used a pretrained deep learning model that could return a skeleton of a human in a video. However, OpenPose came with some huge disadvantages. The installation process was extremely difficult because of dependency issues that were not well documented, the computation time was very slow, and the output was a stick figure where the bones would be instead of a point cloud. After about a week of unsuccessful use of OpenPose, the team switched to the MediaPipe framework by Google which addressed all of these concerns.

2) MediaPipe: Similar to OpenPose, MediaPipe [31] is a deep learning network that returns human pose, but it also returns the pose in the form of a 3 dimensional point cloud in real time. The model is lightweight enough to be run on mobile devices. MediaPipe eliminated the need to custom generate a 3D mesh for the subject. The network has models that generate 3D point clouds for the face, hands, body, hair segmentation, iris tracking, and more in case the project is expanded. Using MediaPipe and OpenCV, a 3D point cloud for the face was generated for every frame of a video.

3) Pose Model: The initial model to generate the 3D point cloud for the face was the MediaPipe Pose Estimation model [31]. The output of this model gives an estimation of 33 landmarks across the entire body, 11 of which being on the face. The origin on the model is centered at the hips of the person in the video. Thus, when the human was at an orientation where their body was sideways relative to the camera frame (side profile), the model attempted to estimate depth of the occluded body parts such as the hidden arm, leg, hip, etc. In addition, if the human in the video was close enough to the camera to the point where most of their body was out of frame, then the model struggled to estimate where the hip origin was. Because of this, a lot of the face landmarks were poorly estimated in depth, and often in X and Y coordinates as well as shown below.

Because of the poor estimations, the homography computations for rotation and translation were poor.

4) FaceMesh Model: The FaceMesh model in MediaPipe [31] was a much better estimator for 3D facial landmarks. The FaceMesh model generated a total of 468 different landmarks, therefore reducing the impact of an outlier landmark when computing the homography parameters.

When parts of the face are occluded, for example in Figure 38, the Face Mesh Model still estimated facial landmarks in 3D. The estimates proved to be much more accurate than those of the Pose Model, and thus generated much more meaningful homography parameters. The facial landmarks also happened to be ordered, meaning the tip of the nose landmark was the same position in the output array in Figure 37 as Figure 38. This detail eliminated the need to compute feature correspondence, thus eliminating additional noise that may have been added to the homography parameters, and therefore the overall system.

Figure 32: Original Orientation of Face

Figure 33: Tilted Orientation of Face

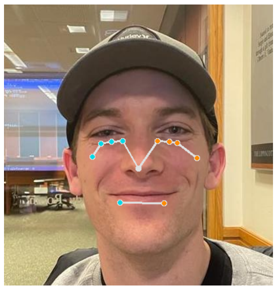


Figure 34: Pose Model Landmarks corresponding to Figure 32

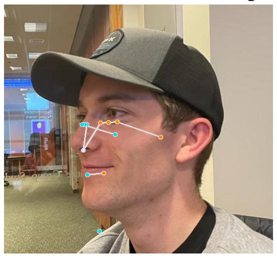


Figure 35: Pose Model Landmarks corresponding to Figure 33

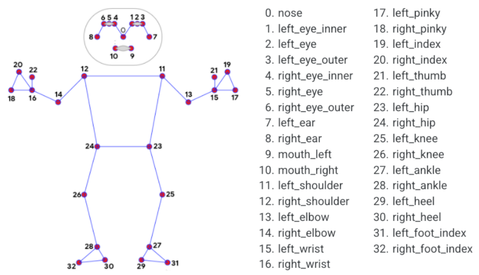


Figure 36: Desired MediaPipe landmarks [31]

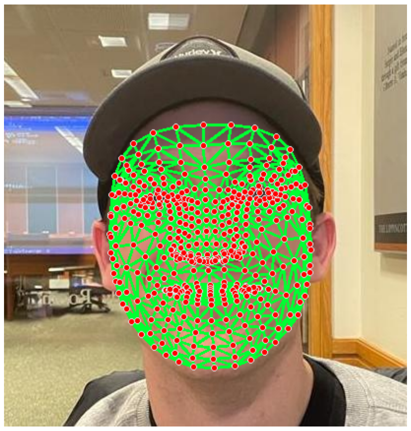


Figure 37: FaceMesh Landmarks corresponding to Figure 32

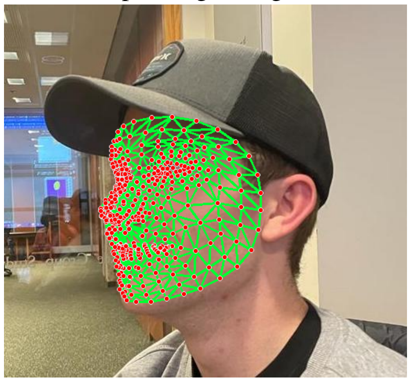


Figure 38: FaceMesh Landmarks corresponding to Figure 33

F. Homography

Since the point clouds are in 3D space, "homography" may be the abused in the sense that it is a 2D transformation. However, since the 3D point cloud was being projected into 2D space, the term homography was used to keep referencing this technique simple.

Given 2 3D point clouds, the homography parameters, being the estimated rotation matrix R and translation vector t, were estimated using the Kabsch algorithm [37] and least-squares minimization fitting [38]. Additionally, the face of the human was assumed to be a rigid body, thus simplifying the computation [39]. Assuming frame i has landmarks A and frame i+1 has landmarks B, the rotation matrix R and tranlation vector t are computed as follows:

$$RA + t = B$$

The first step to estimating the homography parameters was to compute the centroid of the 3D mesh vertices of 2 sequential frames. The centroid is just a mean of all of the points' (N points) coordinates, calculated as follows, where A is the matrix of points coordinates:

$$centroid_A = \frac{1}{N} * \sum_{k=1}^{N} A^k$$

Once the centroids are computed between frame 1 and frame 2, the rotation matrix R is estimated by aligning the centroids and using least squares to get the closest estimate [38], [37]. This eliminated the need to compute translation in the same step, simplifying the calculation. Singular value decomposition (SVD) could then be used on the covariance matrix H to obtain matrices U and V whose columns are the left-singular and right-singular vectors of the singular values in the diagonal matrix S.

$$H = (A - centroid_A) * (B - centroid_B)^T$$

$$[U, S, V] = SVD(H)$$

$$R = V * U^T$$

The translation between the 2 3D point clouds can then be computed using:

$$RA + t = B$$

$$t = B - RA$$

$$t = centroid_B - R * centroid_A$$

G. Pytorch3D Open3D Rendering

The .STL or .OBJ file is warped and rendered using PyTorch3D [40]. PyTorch3D is a tool used to load .OBJ files and store them as a mesh. It has multiple options for rasterizing, shading and rendering the mesh to an image. This library was used to transform the .OBJ file using the homography rotation matrix calculated to track the face of the person. Once the mask was successfully rendered, mesh vertices were selected using Open3D. These points selected on the Open3D mesh were then mapped to the corresponding location on the human face. Pytorch3D was then used again to superimpose the rendered mask onto the human face. The correct translation and scaling were applied in order for the helmet to look realistic when superimposed on the human.

H. Results and discussion

The final result photos and video links are given in the appendix below. We were successfully able to extract the various points of interest from a human face, find the homography between the consecutive frames and apply that homography to a subject 3D mesh (in this case, an Iron Man helmet). We were also able to align the face and the 3D mesh correctly for every frame with appropriate scaling.

I. Future Work

The superimposed helmet was implemented in 2D, resulting in some instances where an ear or a nose would stick out of the helmet. A more robust superimposition can be implemented in 3D, such that the helmet perfectly wraps around the head. Additionally, objects such as gloves and body suit could be also be warped onto the human model instead of only a helmet.

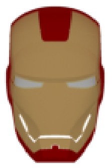


Figure 39: Mask render corresponding to Figure 32 before homography

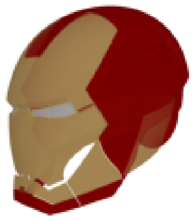


Figure 40: Mask render corresponding to Figure 33 after applied homography parameters

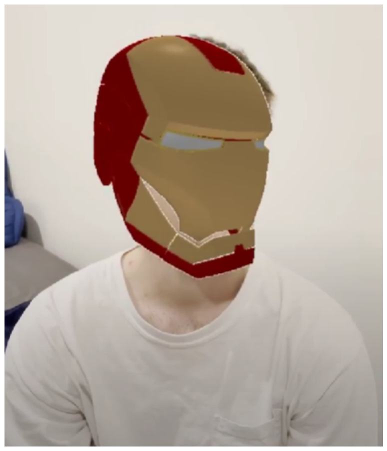


Figure 41: Superimposed output showing rotation

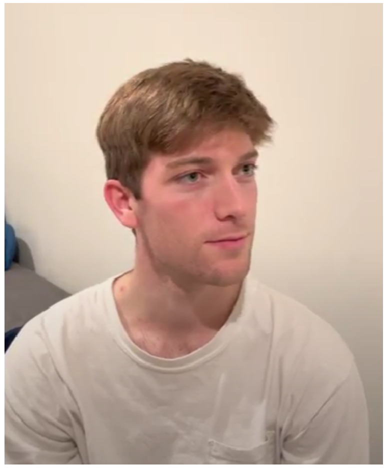


Figure 42: Original image

J. Appendix, Files, Videos

The link to google drive can be found HERE. This drive contains the presentation, presentation video, output videos and final code used in this project. Some of our results are mentioned below.

Acknowledgments

Special shout-out to all of my peers that made my time at Penn one to remember. This experience was enhanced immeasurably by the sharp minds of all of the teammates and friends that I had the pleasure of working with. Thank you!

References

[1] Stanford Artificial Intelligence Laboratory et al., "Robotic operating system." [Online]. Available: https://www.ros.org

[2] "Meam 620 project 1 phase 1 handout."

[3] A. Saalfeld, "Topologically consistent line simplification with the douglas-peucker algorithm," *Cartography and Geographic Information Science*, vol. 26, no. 1, pp. 7–18, 1999.

[4] D. K. Prasad, M. K. Leung, C. Quek, and S.-Y. Cho, "A novel framework for making dominant point detection methods non-parametric," *Image and Vision Computing*, vol. 30, no. 11, pp. 843–859, 2012.

[5] P. Chaudhari, "Ese650 course lecture notes," https://pratikac.github.io/pub/$21_e se650.pdf$, 2022.

[6] L. Drolet, F. Michaud, and J. Côté, "Adaptable sensor fusion using multiple kalman filters," in *Proceedings. 2000 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2000)(Cat. No. 00CH37113)*, vol. 2. IEEE, 2000, pp. 1434–1439.

[7] J. Sola, "Quaternion kinematics for the error-state kalman filter. arxiv 2017," *arXiv preprint arXiv:1711.02508*.

[8] E. Kraft, "A quaternion-based unscented kalman filter for orientation tracking," in *Proceedings of the sixth international conference of information fusion*, vol. 1, no. 1. IEEE Cairns, 2003, pp. 47–54.

[9] R. Cen, T. Jiang, Y. Tan, X. Su, and F. Xue, "A low-cost visual inertial odometry for mobile vehicle based on double stage kalman filter," *Signal Processing*, vol. 197, p. 108537, 2022.

[10] C. Taylor, "Meam620 course lecture notes," https://alliance.seas.upenn.edu/ meam620/wiki/, 2022.

[11] L. Breiman, "Random forests," *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.

[12] M. Kubat, "Neural networks: a comprehensive foundation by simon haykin, macmillan, 1994, isbn 0-02-352781-7." *The Knowledge Engineering Review*, vol. 13, no. 4, pp. 409–412, 1999.

[13] A. I. Mourikis, S. I. Roumeliotis *et al.*, "A multi-state constraint kalman filter for vision-aided inertial navigation." in *ICRA*, vol. 2, 2007, p. 6.

[14] K. Sun, K. Mohta, B. Pfrommer, M. Watterson, S. Liu, Y. Mulgaonkar, C. J. Taylor, and V. Kumar, "Robust stereo visual inertial odometry for fast autonomous flight," *IEEE Robotics and Automation Letters*, vol. 3, no. 2, pp. 965–972, 2018.

[15] M. Bloesch, S. Omari, M. Hutter, and R. Siegwart, "Robust visual inertial odometry using a direct ekf-based approach," in *2015 IEEE/RSJ international conference on intelligent robots and systems (IROS)*. IEEE, 2015, pp. 298–304.

[16] R. G. Valenti, I. Dryanovski, and J. Xiao, "Keeping a good attitude: A quaternion-based orientation filter for imus and margs," *Sensors*, vol. 15, no. 8, pp. 19 302–19 330, 2015.

[17] M. Burri, J. Nikolic, P. Gohl, T. Schneider, J. Rehder, S. Omari, M. W. Achtelik, and R. Siegwart, "The euroc micro aerial vehicle datasets," *The International Journal of Robotics Research*, vol. 35, no. 10, pp. 1157–1163, 2016.

[18] M. R. Maxey and J. J. Riley, "Equation of motion for a small rigid sphere in a nonuniform flow," *The Physics of Fluids*, vol. 26, no. 4, pp. 883–889, 1983.

[19] P. G. Thomasson and C. A. Woolsey, "Vehicle motion in currents," *IEEE Journal of Oceanic Engineering*, vol. 38, no. 2, pp. 226–242, 2013.

[20] A. Talaei and T. J. Garrett, "On the Maxey-Riley equation of motion and its extension to high Reynolds numbers," no. August, 2020. [Online]. Available: http://arxiv.org/abs/2006.16577

[21] D. Gil, "Computation of inertial particle trajectories in steady streaming flows," no. September, 2015.

[22] S. G. Prasath, V. Vasan, and R. Govindarajan, "Accurate solution method for the Maxey-Riley equation, and the effects of Basset history," *Journal of Fluid Mechanics*, vol. 868, pp. 428–460, 2019.

[23] M. Farazmand and G. Haller, "The Maxey–Riley equation: Existence, uniqueness and regularity of solutions," *Nonlinear Analysis: Real World Applications*, vol. 22, pp. 98–106, 2015.

[24] A. Daitche and T. Tél, "Memory effects are relevant for chaotic advection of inertial particles," *Physical Review Letters*, vol. 107, no. 24, pp. 1–5, 2011.

[25] H.-W. Dong, W.-Y. Hsiao, L.-C. Yang, and Y.-H. Yang, "Musegan: Multi-track sequential generative adversarial networks for symbolic music generation and accompaniment," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, no. 1, 2018.

[26] T. Bertin-Mahieux, D. Ellis, B. Whitman, and P. Lamere, "The million song dataset in proceedings of the 12th international society for music information retrieval conference," *Miami, October*, vol. 24, pp. 591–596, 2011.

[27] I. Tham, "Generating music using deep learning."

[28] S. Sulun, M. E. Davies, and P. Viana, "Symbolic music generation conditioned on continuous-valued emotions," *IEEE Access*, vol. 10, pp. 44 617–44 626, 2022.

[29] H.-T. Hung, J. Ching, S. Doh, N. Kim, J. Nam, and Y.-H. Yang, "Emopia: a multi-modal pop piano dataset for emotion recognition and emotion-based music generation," *arXiv preprint arXiv:2108.01374*, 2021.

[30] L. N. Ferreira and J. Whitehead, "Learning to generate music with sentiment," *arXiv preprint arXiv:2103.06125*, 2021.

[31] C. Lugaresi, J. Tang, H. Nash, C. McClanahan, E. Uboweja, M. Hays, F. Zhang, C.-L. Chang, M. G. Yong, J. Lee, W.-T. Chang, W. Hua, M. Georg, and M. Grundmann, "Mediapipe: A framework for building perception pipelines," 2019.

[32] Z. Cao, G. Hidalgo, T. Simon, S.-E. Wei, and Y. Sheikh, "Openpose: realtime multi-person 2d pose estimation using part affinity fields," *IEEE transactions on pattern analysis and machine intelligence*, vol. 43, no. 1, pp. 172–186, 2019.

[33] Z. Gojcic, O. Litany, A. Wieser, L. J. Guibas, and T. Birdal, "Weakly supervised learning of rigid 3d scene flow," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 5692–5703.

[34] G. Yang and D. Ramanan, "Upgrading optical flow to 3d scene flow through optical expansion," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 1334–1343.

[35] P. Scovanner, S. Ali, and M. Shah, "A 3-dimensional sift descriptor and its application to action recognition," in *Proceedings of the 15th ACM International Conference on Multimedia*, ser. MM '07. New York, NY, USA: Association for Computing Machinery, 2007, p. 357–360. [Online]. Available: https://doi.org/10.1145/1291233.1291311

[36] M. Tancik, P. P. Srinivasan, B. Mildenhall, S. Fridovich-Keil, N. Raghavan, U. Singhal, R. Ramamoorthi, J. T. Barron, and R. Ng, "Fourier features let networks learn high frequency functions in low dimensional domains," *NeurIPS*, 2020.

[37] W. Kabsch, "A solution for the best rotation to relate two sets of vectors," *Acta Crystallographica Section A: Crystal Physics, Diffraction, Theoretical and General Crystallography*, vol. 32, no. 5, pp. 922–923, 1976.

[38] K. S. Arun, T. S. Huang, and S. D. Blostein, "Least-squares fitting of two 3-d point sets," *IEEE Transactions on pattern analysis and machine intelligence*, no. 5, pp. 698–700, 1987.

[39] "Finding optimal rotation and translation between corresponding 3d points," http://nghiaho.com/?page_id=671, accessed: 2020-11-16.

[40] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds. Curran Associates, Inc., 2019, pp. 8024–8035. [Online]. Available: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf