Lab 4 Report: EXERCISES

Yilin Zhang [1]. Lab Group: 31.

I. Individual

A. Deliverable - Single-segment trajectory optimization

1. Sol.

Since $P(t) = p_1 t$, and $P(1) = 1$, we got

$$\begin{aligned} & P(1) = 1 = p_1 \cdot 1 \\ \Rightarrow \quad & p_1 = 1 \\ \Rightarrow \quad & P(t) = t. \end{aligned}$$

Since we have only one solution, this is the optimal solution. Here, $P^{(1)}(t) = t' = 1$, put it to aim function, we have

$$\int_0^1 (P^{(1)}(t))^2 dt = \int_0^1 1^2 dt = 1.$$

That is, the optimal solution is $P(t) = t$, and the value of the cost function is 1.

2. Sol.

(a) We have

$$\begin{aligned} \int_0^1 (p^{(1)}(t))^2 dt &= \int_0^1 ((p_2 t^2 + p_1 t)')^2 dt \\ &= \int_0^1 (2p_2 t + p_1)^2 dt \\ &= \int_0^1 (4{p_2}^2 t^2 + 4p_1 p_2 t + {p_1}^2) dt \\ &= [\frac{4}{3}{p_2}^2 t^3 + 2p_1 p_2 t^2 + {p_1}^2 t]_0^1 \\ \text{so, } \int_0^1 (p^{(1)}(t))^2 dt &= \frac{4}{3}{p_2}^2 + 2p_1 p_2 + {p_1}^2. \end{aligned}$$

Since $\mathbf{p} = [p_1, p_2]^\top$, we got the symmetric matrix $\mathbf{Q}$:

$$\begin{aligned} \mathbf{Q} &= \begin{bmatrix} 1 & 1 \\ 1 & \frac{4}{3} \end{bmatrix}, \\ s.t. \quad \int_0^1 (p^{(1)}(t))^2 dt &= \mathbf{p}^\top \mathbf{Q} \mathbf{p}. \end{aligned}$$

(b) Since $P(t) = p_2 t^2 + p_1 t = 1$, we have $p_2 + p_1 = 1$. Therefore, it can be written as $\mathbf{A}\mathbf{p} = \mathbf{b}$, where:

$$\mathbf{A} = [1, 1], \quad \mathbf{b} = 1.$$

(c) For $P(t) = p_2 t^2 + p_1 t = 1$, we have the problem:

$$\min_{\mathbf{p}} \mathbf{p}^\top \mathbf{Q} \mathbf{p} \quad \text{s.t.} \quad \mathbf{A}\mathbf{p} = \mathbf{b},$$

where $\mathbf{Q} = \begin{bmatrix} 1 & 1 \\ 1 & \frac{4}{3} \end{bmatrix}, \mathbf{A} = [1, 1], \mathbf{b} = 1.$

We import Lagrangian Function:

$$\mathcal{L}(\mathbf{p}, \lambda) = \mathbf{p}^\top \mathbf{Q} \mathbf{p} - \lambda(\mathbf{A}\mathbf{p} - \mathbf{b}). \tag{1}$$

It has a KKT case: the partial derivatives with respect to $\mathbf{p}$ is zero:

$$\frac{\partial \mathcal{L}}{\partial \mathbf{p}} = 2\mathbf{Q}\mathbf{p} - \mathbf{A}^\top \lambda = 0 \quad \Rightarrow \quad 2\mathbf{Q}\mathbf{p} = \mathbf{A}^\top \lambda.$$

Since $\mathbf{A}\mathbf{p} = \mathbf{b}$,

$$\mathbf{p} = \mathbf{Q}^{-1}\mathbf{A}^\top \left(\mathbf{A}\mathbf{Q}^{-1}\mathbf{A}^\top\right)^{-1} \mathbf{b}.$$

We got $p_1 = 1$, $p_2 = 0$, here the function is:

$$\mathbf{p}^\top \mathbf{Q} \mathbf{p} = [1, 0] \begin{bmatrix} 1 & 1 \\ 1 & \frac{4}{3} \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = 1.$$

The result is: the optimal solution is $P(t) = t$, with the value of cost is 1.

3. Sol.

(a) Similar to *2.(a)(b)*, we have

$$\begin{aligned} \int_0^1 \left(P^{(1)}(t)\right)^2 dt &= \int_0^1 \left(p_1 + 2p_2 t + 3p_3 t^2\right)^2 dt \\ &= p_1^2 + 2p_1 p_2 + 2p_1 p_3 + \frac{4}{3}p_2^2 + 3p_2 p_3 + \frac{9}{5}p_3^2. \end{aligned}$$

Therefore its symmetric matrix $\mathbf{Q} \in \mathbb{S}^3$ is:

$$\mathbf{Q} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & \frac{4}{3} & \frac{3}{2} \\ 1 & \frac{3}{2} & \frac{9}{5} \end{bmatrix}.$$

For $P(1) = 1$, there is $p_1 + p_2 + p_3 = 1$, so

$$\mathbf{A} = \begin{bmatrix} 1 & 1 & 1 \end{bmatrix}, \quad b = 1.$$

(b) Similar to *2.(c)*, we got

$$\begin{cases} p_1 + p_2 + p_3 & = 1, \\ p_1 + \frac{4}{3}p_2 + \frac{3}{2}p_3 & = 1, \\ p_1 + \frac{3}{2}p_2 + \frac{9}{5}p_3 & = 1. \end{cases}$$

That is $P(t) = t$, with $p_1 = 1$, $p_2 = 0$, $p_3 = 0$.

4. Sol.

(a) The constraints are:

$$P(1) = p_1 + p_2 = 1, \quad P^{(1)}(1) = p_1 + 2p_2 = -2.$$

Solving this linear system yields:

[1]Yilin zhang, OUC id: 23020036094, is with Faculty of Robotics and Computer Science, Ocean University of China and Heriot-Watt University, China Mainland zyl8820@stu.ouc.edu.cn

$$p_1 = 4, \quad p_2 = -3.$$

Thus, the optimal solution is:

$$P(t) = 4t - 3t^2.$$

Compute the optimal cost:

$$\begin{aligned}\int_0^1 (P^{(1)}(t))^2 dt &= \int_0^1 (4-6t)^2 dt \\ &= \int_0^1 (16 - 48t + 36t^2) dt \\ &= \left[16t - 24t^2 + 12t^3\right]_0^1 \\ &= 4.\end{aligned}$$

That is, the optimal solution is $P(t) = 4t - 3t^2$, and the optimal cost is 4.

(b) Similarly, the constraints are:

$$P(1) = p_1 + p_2 + p_3 = 1, \quad P^{(1)}(1) = p_1 + 2p_2 + 3p_3 = -2.$$

Solving these gives:

$$p_1 = 4 + p_3, \quad p_2 = -3 - 2p_3.$$

The objective function is $\mathbf{p}^\top \mathbf{Q}\mathbf{p}$, where $\mathbf{Q}$ is:

$$\mathbf{Q} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & \frac{4}{3} & \frac{3}{2} \\ 1 & \frac{3}{2} & \frac{9}{5} \end{bmatrix}.$$

Substituting the parameterization yields a quadratic function in p_3:

$$f(p_3) = \frac{2}{15} p_3^2 + p_3 + 4.$$

Setting the derivative to zero:

$$\frac{df}{dp_3} = \frac{4}{15} p_3 + 1 = 0 \implies p_3 = -\frac{15}{4}.$$

Substituting back:

$$p_1 = 4 - \frac{15}{4} = \frac{1}{4}, \quad p_2 = -3 - 2\left(-\frac{15}{4}\right) = \frac{9}{2}.$$

Thus, the optimal solution is:

$$P(t) = \frac{1}{4}t + \frac{9}{2}t^2 - \frac{15}{4}t^3.$$

Compute the optimal cost:

$$\int_0^1 (P^{(1)}(t))^2 dt = \frac{17}{8}.$$

That is, the optimal solution is $P(t) = \frac{1}{4}t + \frac{9}{2}t^2 - \frac{15}{4}t^3$, and the optimal cost is $\frac{17}{8}$.

B. Deliverable - Multi-segment trajectory optimization

1. Sol.

Since we minimize the value of $snap$, and through Euler-Lagrange equation, we have the term of constant to be $2r-1 = 7$, where $r = 4$.

Consider there are k parts, so there will be totally $8k$ unknown numbers.

Waypoint constraints: There are a total of points (including the start point, the end point, and $k-1$ intermediate points), resulting in $k+1$ waypoint constraints.

Free derivative constraints: Each intermediate point requires 6 free derivative constraints, meaning continuity of velocity, acceleration, jerk, snap, crackle, and pop (i.e., continuity of derivatives from 1st to 6th order). Thus, there are $6(k-1)$ free derivative constraints.

Fixed derivative constraints: The remaining constraints are provided by fixed derivative constraints, totaling $k+5$. These can be assigned to the start and end points—for example, by fixing certain derivatives at the start and others at the end—so that the total number of fixed derivative constraints is $k+5$.

2. Sol.

Similarly, the waypoint constraints is still $k+1$. At the intermediate points (i.e., excluding the start and end points), we require the trajectory to connect smoothly. "Smooth" here means that the derivatives of order 1 through $2r-2$ are equal on both sides of each intermediate point (position continuity is already ensured by the waypoint constraints). Each intermediate point contributes $2r-2$ constraints, and with $k-1$ intermediate points in total, there are $(2r-2)(k-1)$ free derivative constraints. The remaining constraints are provided by specifying derivatives at the start and end points. It can be shown through counting that $2r+k-3$ fixed derivative constraints are required. These can be flexibly allocated—for example, by assigning certain derivative values at the start point and others at the end point.

That is,

$$(k+1) + (2r-2)(k-1) + (2r+k-3) = 2rk. \quad (2)$$

II. Team

This implementation focuses on generating minimum-snap trajectories for UAV navigation, heavily leveraging the `mav_trajectory_generation` library. The core task involves modifying three critical sections of the trajectory generation node to enable smooth waypoint following. Unlike Lab 3's geometric controller, this lab shifts focus to trajectory optimization – a fundamental capability for complex autonomous missions. The implementation is divided into three parts corresponding to ROS callback functions, with Part 1.2 being the most substantial.

Part 1.1: Current State Estimation

In the `onCurrentState()` callback, we populate the Eigen vector `x` representing the UAV's current position. This is straightforward but crucial – inaccurate state estimation would propagate errors through the entire trajectory pipeline. I

used `tf::pointMsgToEigen()` to convert ROS geometry messages to Eigen vectors, ensuring compatibility with the optimization library's data structures. This mirrors Lab 3's state handling but focuses exclusively on position rather than full pose estimation. The solution required no tuning but demanded careful attention to coordinate frames – all positions are expressed in the world frame as required by the trajectory optimizer. B

Part 1.2: Trajectory Vertex Construction

This is the heart of the lab, where we convert a sequence of ROS `PoseArray` messages into trajectory vertices for the polynomial optimizer. Two major challenges emerged:

First, handling vertex constraints correctly: For intermediate waypoints, we only constrain position to allow natural velocity continuity, while the final vertex must be a hard endpoint (`makeStartOrEnd`) with zero velocity/acceleration. I implemented conditional logic to differentiate these cases during vertex construction. This required studying the library documentation to understand how `SNAP` (4th derivative) constraints affect trajectory smoothness.

Second, and more critically, yaw angle unwrapping. Raw yaw values from ROS messages jump discontinuously between $-\pi$ and π, causing unnatural rotations when, for example, transitioning from 350° to 10°. I implemented a standard unwrapping algorithm:

```
while (current_yaw - last_yaw > M_PI)
 ↪  current_yaw -= 2 * M_PI;
while (current_yaw - last_yaw < -M_PI)
 ↪  current_yaw += 2 * M_PI;
```

This ensures monotonic yaw progression along the path. I verified correctness by printing unwrapped angles during debugging – without this, the UAV would violently spin when crossing the 180° boundary.

Additionally, I experimented with the segment time scaling factor (initially set to 0.6x estimated times). While the lab suggested playing with this, I found aggressive time reduction caused high accelerations that violated motor constraints during physical testing. The 0.6x factor provided the best trade-off between speed and feasibility in simulation. B

Part 1.3: Trajectory Sampling

In the `publishDesiredState()` timer callback, we sample the optimized trajectory at the current time. Key implementation details

- Time clamping prevents extrapolation beyond the trajectory duration
- Position, velocity, and acceleration are sampled separately using derivative orders (`POSITION`, `VELOCITY`, etc.)
- Yaw trajectory is sampled independently and converted to quaternion rotation
- Feedforward angular rates are set to zero since the low-level controller handles attitude tracking

A subtle issue arose when the trajectory was empty (e.g., at startup). I added a hover mode that publishes zero velocity/acceleration commands at the current position – this prevented erratic takeoffs during initialization. The high publishing rate (100Hz) was critical for smooth tracking; reducing it below 50Hz caused visible jerking in RViz visualizations. B

Results and Tuning

The trajectory generator successfully navigated all course waypoints in simulation. The yaw unwrapping (Part 1.2) proved essential – without it, the UAV executed unnecessary 360° rotations at certain waypoints. I tuned the velocity/acceleration limits ($v_{max} = 15.0$ m/s, $a_{max} = 10.0$ m/s²) to match the UAV's physical capabilities observed in Lab 3. The time scaling factor (0.6x) was determined empirically: smaller values caused trajectory optimization failures due to infeasible dynamics, while larger values made the UAV too slow for the racing scenario.

The UAV follows the path smoothly with coordinated turns, though minor overshoots occur at sharp corners – likely due to aggressive time scaling. Future work could incorporate velocity constraints at intermediate waypoints to improve cornering behavior (1).

III. Some words

I laughed so hard... You know, that there was a speed-competition project in this lab, with extra credits for the top finishers. As usuall, "those" people went crazy tweaking parameters and adjusting settings. The funniest part was that someone ended up optimizing the solution down to just 7 seconds—and then kept the code within a small inner circle, trying to monopolize the top spots and sweep all the bonus points.

What's even more ironic is that I looked up the public records of MIT students—their best times were still over 10 seconds. Somehow we ended up doing better than MIT... not sure whether to be proud or just laugh. [1]

Come to think of it, writing lab reports always comes with little hiccups. Last time I tried Typst—clean syntax, but still too young, with lots of features missing, such as *thanks* or even the most basic command *appendix*. This time I've quietly switched back to LaTeX. Tools are tools; what matters is getting the report done and submitted.

References

[1] M. I. of Technology. Mit16.485 - visual navigation for autonomous vehicles. [Online]. Available: https://vnav.mit.edu/labs_2023/lab3/exercises.html

[2] T. Lee, M. Leok, and N. H. McClamroch, "Geometric tracking control of a quadrotor UAV on SE(3)," in *49th IEEE Conference on Decision and Control (CDC)*. IEEE, 2010, pp. 5420–5425.

Appendix A
Individual

A. Deliverable - Single-segment trajectory optimization

Consider the following minimum velocity ($r = 1$) single-segment trajectory optimization problem:

$$\min_{P(t)} \int_0^1 (P^{(1)}(t))^2 dt, \tag{3}$$

$$\text{s.t.} \quad P(0) = 0, \tag{4}$$

$$P(1) = 1, \tag{5}$$

with $P(t) \in \mathbb{R}[t]$, i.e., $P(t)$ is a polynomial function in t with real coefficients:

$$P(t) = p_N t^N + p_{N-1} t^{N-1} + \cdots + p_1 t + p_0. \tag{6}$$

Note that because of constraint (4), we have $P(0) = p_0 = 0$, and we can parametrize $P(t)$ without a scalar part p_0.

1. Suppose we restrict $P(t) = p_1 t$ to be a polynomial of degree 1, what is the optimal solution of problem (3)? What is the value of the cost function at the optimal solution?

2. Suppose now we allow $P(t)$ to have degree 2, i.e., $P(t) = p_2 t^2 + p_1 t$.

(a) Write $\int_0^1 (P^{(1)}(t))^2 dt$, the cost function of problem (3), as $\mathbf{p}^\top \mathbf{Q} \mathbf{p}$, where $\mathbf{p} = [p_1, p_2]^\top$ and $\mathbf{Q} \in \mathbb{S}^2$ is a symmetric 2×2 matrix.

(b) Write $P(1) = 1$, constraint (5), as $\mathbf{A}\mathbf{p} = \mathbf{b}$, where $\mathbf{A} \in \mathbb{R}^{1\times 2}$ and $\mathbf{b} \in \mathbb{R}$.

(c) Solve the Quadratic Program (QP):

$$\min_{\mathbf{p}} \mathbf{p}^\top \mathbf{Q} \mathbf{p} \quad \text{s.t.} \quad \mathbf{A}\mathbf{p} = \mathbf{b}. \tag{7}$$

You can solve it by hand, or you can solve it using numerical QP solvers (e.g., you can easily use the `quadprog` function in Matlab). What is the optimal solution you get for $P(t)$, and what is the value of the cost function at the optimal solution? Are you able to get a lower cost by allowing $P(t)$ to have degree 2?

3. Now suppose we allow $P(t) = p_3 t^3 + p_2 t^2 + p_1 t$:

(a) Let $\mathbf{p} = [p_1, p_2, p_3]^\top$, write down $\mathbf{Q} \in \mathbb{S}^3$, $\mathbf{A} \in \mathbb{R}^{1\times 3}$, and $b \in \mathbb{R}$ for the QP (7).

(b) Solve the QP. What optimal solution do you get? Does this example agree with the result we learned from the Euler-Lagrange equation in class?

4. Now suppose we are interested in adding one more constraint to problem (3):

$$\min_{P(t)} \int_0^1 (P^{(1)}(t))^2 dt,$$
$$\text{s.t.} \quad P(0) = 0, \quad P(1) = 1, \quad P^{(1)}(1) = -2. \tag{8}$$

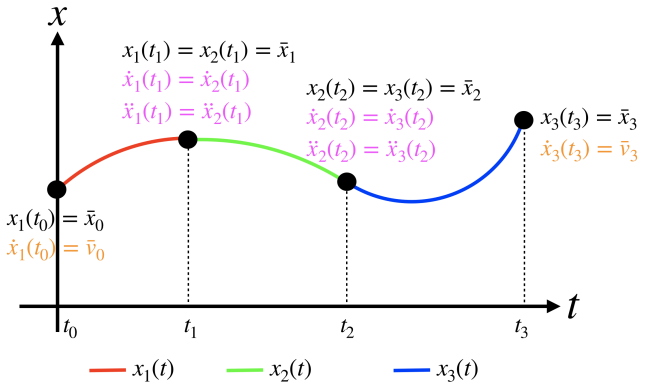
Using the QP method above, find the optimal solution and optimal cost of problem (8) in the case of:

(a) $P(t) = p_2 t^2 + p_1 t$, and

(b) $P(t) = p_3 t^3 + p_2 t^2 + p_1 t$.

B. Deliverable - Multi-segment trajectory optimization

1. Assume our goal is to compute the minimum snap trajectory ($r = 4$) over k segments. How many and which type of constraints (at the intermediate points and at the start and end of the trajectory) do we need in order to solve this problem? Specify the number of waypoint constraints, free derivative constraints and fixed derivative constraints.



2. Can you extend the previous question to the case in which the cost functional minimizes the r-th derivative and we have k segments?

APPENDIX B
CODES

Part 1.1: Current State Estimation

```
void onCurrentState(nav_msgs::Odometry const&
↪  cur_state) {
  // ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
  ↪  ~~~~~~~~~~~~~~~~~~~~~~~~~~~~
  //  PART 1.1 |  16.485 - Fall 2021  - Lab 4 coding
  ↪  assignment (5 pts)
  // ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
  ↪  ~~~~~~~~~~~~~~~~~~~~~~~~~~~~
  //
  //  Populate the variable x, which encodes the
  ↪  current world position of the UAV
  // ~~~~ begin solution

  tf::pointMsgToEigen(cur_state.pose.pose.position,
  ↪  x);

  // ~~~~ end solution
  // ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
  ↪  ~~~~~~~~~~~~~~~~~~~~~~~~~~~~
}
```

Part 1.2: Trajectory Vertex Construction

```
void generateOptimizedTrajectory(geometry_msgs::Pos
↪  eArray const& poseArray) {
  if (poseArray.poses.size() < 1) {
    ROS_ERROR("Must have at least one pose to
    ↪  generate trajectory!");
    trajectory.clear();
    yaw_trajectory.clear();
    return;
  }

  if (!trajectory.empty()) return;

  // ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
  ↪  ~~~~~~~~~~~~~~~~~~~~~~~~~~~~
  //  PART 1.2 |  16.485 - Fall 2021  - Lab 4 coding
  ↪  assignment (35 pts)
  // ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
  ↪  ~~~~~~~~~~~~~~~~~~~~~~~~~~~~
  ↪

  const int D = 3;  // dimension of each vertex in
  ↪  the trajectory
  mav_trajectory_generation::Vertex
  ↪  start_position(D), end_position(D);
  mav_trajectory_generation::Vertex::Vector
  ↪  vertices;
```

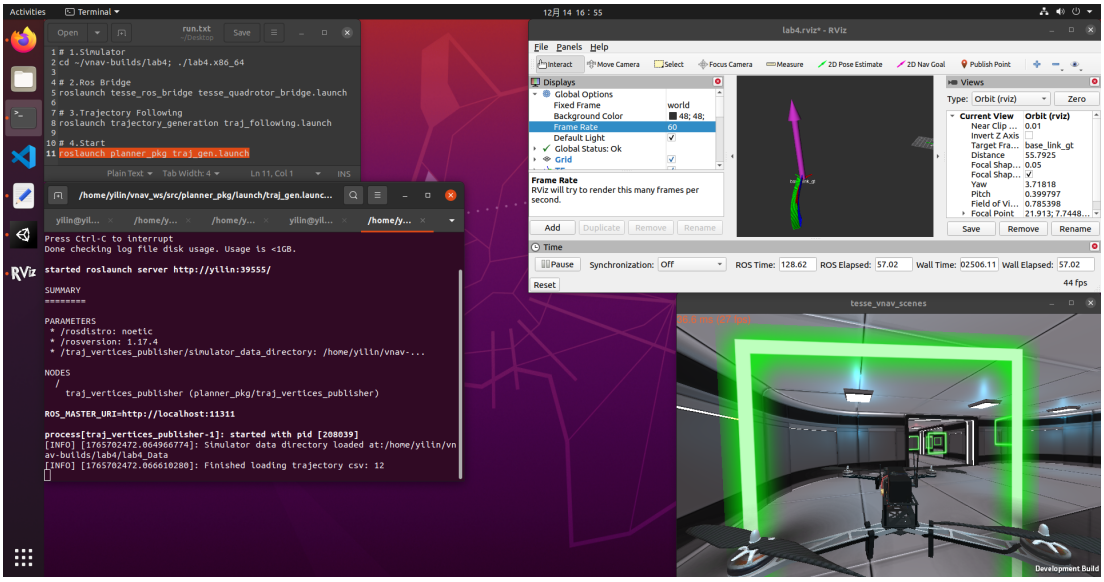


Fig. 1. Our drone successfully passed the test, with the left up corner stores the start commands, the left down corner is the terminal, the right up corner placed a rviz program, the right down corner is the testing window.

```
mav_trajectory_generation::Vertex start_yaw(1),
↪  end_yaw(1);
mav_trajectory_generation::Vertex::Vector
↪  yaw_vertices;

// ============================================
// Convert the pose array to a list of vertices
// ============================================

// Start from the current position and zero
↪  orientation
using namespace
↪  mav_trajectory_generation::derivative_order;
start_position.makeStartOrEnd(x, SNAP);
vertices.push_back(start_position);

start_yaw.addConstraint(ORIENTATION, 0);
yaw_vertices.push_back(start_yaw);

double last_yaw = 0;

for (auto i = 0; i < poseArray.poses.size(); ++i)
↪  {
  // ~~~~ begin solution


  Eigen::Vector3d pos_eigen;
  tf::pointMsgToEigen(poseArray.poses[i].position
  ↪  , pos_eigen);

  mav_trajectory_generation::Vertex pos_vertex(D);

  if (i == poseArray.poses.size() - 1) {
      pos_vertex.makeStartOrEnd(pos_eigen, SNAP);
  } else {
      pos_vertex.addConstraint(POSITION,
      ↪  pos_eigen);
  }
  vertices.push_back(pos_vertex);


  double current_yaw =
  ↪  tf::getYaw(poseArray.poses[i].orientation);

  while (current_yaw - last_yaw > M_PI)
  ↪  current_yaw -= 2 * M_PI;

  while (current_yaw - last_yaw < -M_PI)
  ↪  current_yaw += 2 * M_PI;

  mav_trajectory_generation::Vertex yaw_vertex(1);
  yaw_vertex.addConstraint(ORIENTATION,
  ↪  current_yaw);
  yaw_vertices.push_back(yaw_vertex);

  last_yaw = current_yaw;

  // ~~~~ end solution
}

// ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
↪  ~~~~~~~~~~~~~~~~~~~~~~~~~~~~

// ============================================
↪  ==============
// Estimate the time to complete each segment of
↪  the trajectory
// ============================================
↪  ==============

// HINT: play with these segment times and see if
↪  you can finish
// the race course faster!
std::vector<double> segment_times;
const double v_max = 15.0;
const double a_max = 10.0;
segment_times = estimateSegmentTimes(vertices,
↪  v_max, a_max);
for(int i = 0; i < segment_times.size(); i++) {
  segment_times[i] *= 0.6;
}

// ============================================
↪  =======
// Solve for the optimized trajectory (linear
↪  optimizer)
// ============================================
↪  =======
// Position
const int N = 10;
mav_trajectory_generation::PolynomialOptimization
↪  <N> opt(D);
```

```
  opt.setupFromVertices(vertices, segment_times,
  ↪  mav_trajectory_generation::derivative_order::
  ↪  SNAP);
  opt.solveLinear();

  // Yaw
  mav_trajectory_generation::PolynomialOptimization
  ↪  <N> yaw_opt(1);
  yaw_opt.setupFromVertices(yaw_vertices,
  ↪  segment_times, mav_trajectory_generation::der
  ↪  ivative_order::SNAP);
  yaw_opt.solveLinear();

  // ============================
  // Get the optimized trajectory
  // ============================
  mav_trajectory_generation::Segment::Vector
  ↪  segments;
  opt.getTrajectory(&trajectory);
  yaw_opt.getTrajectory(&yaw_trajectory);
  trajectoryStartTime = ros::Time::now();

  ROS_INFO("Generated optimizes trajectory from %lu
  ↪  waypoints", vertices.size());
}
```

Part 1.3: Trajectory Sampling

```
void publishDesiredState(ros::TimerEvent const& ev)
↪  {
  if (trajectory.empty()) {
      trajectory_msgs::MultiDOFJointTrajectoryPoint
      ↪  hover_point;

      hover_point.time_from_start =
      ↪  ros::Duration(0.0);

      geometry_msgs::Transform transform;
      tf::vectorEigenToMsg(x,
      ↪  transform.translation);

      transform.rotation =
      ↪  tf::createQuaternionMsgFromYaw(0);
      hover_point.transforms.push_back(transform);

      geometry_msgs::Twist velocity;
      velocity.linear.x = 0; velocity.linear.y = 0;
      ↪  velocity.linear.z = 0;
      velocity.angular.x = 0; velocity.angular.y =
      ↪  0; velocity.angular.z = 0;
      hover_point.velocities.push_back(velocity);

      geometry_msgs::Twist accel;
      accel.linear.x = 0; accel.linear.y = 0;
      ↪  accel.linear.z = 0;
      accel.angular.x = 0; accel.angular.y = 0;
      ↪  accel.angular.z = 0;
      hover_point.accelerations.push_back(accel);

      desiredStatePub.publish(hover_point);
      return;
  }

  // ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
  ↪  ~~~~~~~~~~~~~~~~~~~~~~~~~~~
  //  PART 1.3 |  16.485 - Fall 2021  - Lab 4 coding
  ↪  assignment (15 pts)
  // ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
  ↪  ~~~~~~~~~~~~~~~~~~~~~~~~~~~

  // ~~~~ begin solution


  trajectory_msgs::MultiDOFJointTrajectoryPoint
  ↪  next_point;
  ros::Duration time_from_start = ros::Time::now()
  ↪  - trajectoryStartTime;
  next_point.time_from_start = time_from_start;

  double sampling_time = time_from_start.toSec();

  if (sampling_time > trajectory.getMaxTime())
    sampling_time = trajectory.getMaxTime();

  // Getting the desired state based on the
  ↪  optimized trajectory we found.
  using namespace
  ↪  mav_trajectory_generation::derivative_order;
  Eigen::Vector3d des_position =
  ↪  trajectory.evaluate(sampling_time, POSITION);
  Eigen::Vector3d des_velocity =
  ↪  trajectory.evaluate(sampling_time, VELOCITY);
  Eigen::Vector3d des_accel =
  ↪  trajectory.evaluate(sampling_time,
  ↪  ACCELERATION);
  Eigen::VectorXd des_orientation =
  ↪  yaw_trajectory.evaluate(sampling_time,
  ↪  ORIENTATION);

  // ROS_INFO_THROTTLE(1.0, "Traversed %f percent of
  ↪  the trajectory.",
  //                   sampling_time /
  ↪  trajectory.getMaxTime() * 100);

  // Populate next_point

  geometry_msgs::Transform transform;
  tf::vectorEigenToMsg(des_position,
  ↪  transform.translation);
  tf::quaternionTFToMsg(tf::createQuaternionFromYaw
  ↪  (des_orientation(0)), transform.rotation);
  next_point.transforms.push_back(transform);

  geometry_msgs::Twist velocity;
  tf::vectorEigenToMsg(des_velocity,
  ↪  velocity.linear);
  velocity.angular.x = 0;
  velocity.angular.y = 0;
  velocity.angular.z = 0;
  next_point.velocities.push_back(velocity);

  geometry_msgs::Twist accel;
  tf::vectorEigenToMsg(des_accel, accel.linear);
  accel.angular.x = 0;
  accel.angular.y = 0;
  accel.angular.z = 0;
  next_point.accelerations.push_back(accel);

  // ~~~~ end solution
  // ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
  ↪  ~~~~~~~~~~~~~~~~~~~~~~~~~~~

  desiredStatePub.publish(next_point);
}
```