Self-Driving Car Using Sequential Quadratic Programming and Interior-Point Methods

Fernando Palafox Escobedo[*]

[*] The University of Texas at Austin
fernandopalafox@utexas.edu

I. Introduction

My research is focused on understanding how to develop autonomous systems of all kinds. A self-driving car is a kind of autonomous system that has the potential for a large-scale impact on the world in the next couple of decades, and I have spent a great deal of time thinking of how these cars would work. However, although I greatly enjoy the theoretical side of my research, it is often good to calibrate the "thinking" with more "doing" in the form of practical implementations with concrete results. In this investigation, I present a practical implementation of the following guiding question: how do you program a self-driving car to go from point A to point B? In the past I have looked into answering this question through the use of sampling-based methods such as rapidly-exploring random trees (RRT) [10]. This time I want to solve it using the tools introduced by the field of numerical optimization. I begin by presenting a formal description of the problem and then provide relevant theoretical concepts, ending each theory section with concrete implementation in the form of an algorithm. Next, I present a set of simulations aimed at illustrating how the theory applies in practice - Figure 1 provides a taste of the final results. The entire code-base with implementations and examples are also provided in the form of a GitHub repository [12]. Finally, I provide a brief discussion of what it all means, and where potential areas of future work lie.

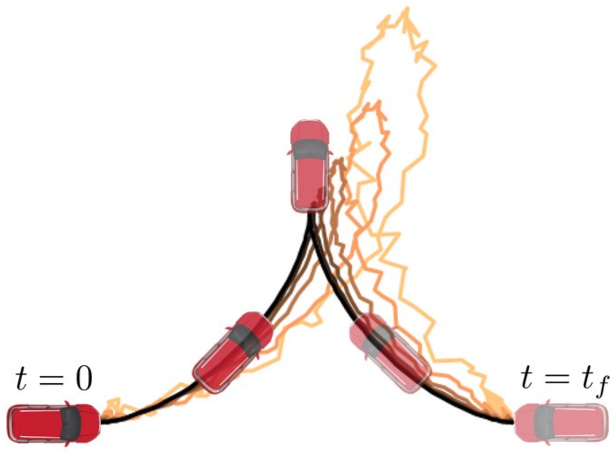

Fig. 1: Optimal trajectory planning for a self-driving car attempting to perform a 180-degree turn. Initial guess in yellow. Final trajectory in black, obtained after 6 iterations of an SQP algorithm.

II. Overview of Investigation & Existing Literature

The theoretical development in this investigation closely follows Nocedal and Wright's, *Numerical Optimization* [11], as it was the primary source of information. However, I made specific efforts to explain any concepts in the way I personally found them to be the most clear when first trying to understand them. This resulted in some additional explanations and small changes in notation. In the interest of brevity and because of the practical nature of the investigation, I did not include any convergence proofs or analysis of theoretical performance. The goal for this investigation was a concrete example of a solution to the guiding question, and as they say, "the proof is in the pudding."

I began this journey by looking into a tractable formulation of the guiding question as outlined in Section III. I defined the question in terms of an optimal-control problem and expressed it as a set of nonlinear program. To solve these, I looked into sequential quadratic programming (SQP), a technique for solving nonlinear constrained optimization problems, following the suggestion of my PhD advisor, Dr. David Fridovich-Keil. SQP is a well-studied concept used in popular optimization libraries such as KNITRO [3] and SNOPT [5]. Reviewing some of the existing literature, I was able to find an instance of a tangential application of SQP in the context of self-driving cars in [4], as well as a brief mention of it for an optimal-control problem in [2]. However, practical applications seemed to mostly be focused on robotic manipulators as in [9], [8], and [13]. Newton's method for solving nonlinear equations is essential to the development of SQP. Therefore, I have included a brief review of its basics in Section IV and an implementation in Algorithm 1. I then go over the theoretical development of SQP in Section V and provide Algorithms 2 and 3, which form the basis of the implementations used to generate the results in Section VIII.

As part of the development of SQP, I found myself in need of another algorithm capable of solving quadratic problems with inequality constraints. For this, I looked into interior-point methods, a powerful set of techniques commonly used in large-scale nonlinear programming. Interior-point methods are notably used in the optimization software package MOSEK [1] and have been applied in a variety of contexts such as power system problems [6], portfolio optimization problems [7], and model-predictive control [14]. In the context of this investigation, they were only required for solving a relatively simple quadratic problem, so only the most basic implementation was necessary.

Implementation of interesting new techniques outlined in literature is commonly bottlenecked by the search of simple, yet non-obvious, details that are crucial for successful execution. Most of the time, this can all be avoided if only the authors include an extra section or appendix listing them out. Although I recognize that this practice may not be applicable to all of the published literature, I still would like to see it become more widespread and in Section VII I have included some of the notable details it took to implement Algorithm 3 and Algorithm 4. I have also included a link to the public GitHub repository containing all of the code.

Finally, I present a set of demonstrations in Section VIII and go over their meaning and potential areas of future research in Section IX. Conducting this investigation was extremely useful to me as an exercise in deep understanding of the concepts I presented. Hopefully it may also be helpful to the reader.

III. Problem Formulation

We begin by modeling the self-driving car as a simple unicycle with the following dynamics

$$\dot{x} = v\cos\theta \tag{1a}$$

$$\dot{y} = v\sin\theta \tag{1b}$$

$$\dot{\theta} = u_1 \tag{1c}$$

$$\dot{v} = u_2 \tag{1d}$$

with state vector $\mathbf{x} = (x, y, \theta, v)$. Given this model, the guiding question can be framed as solving an optimal control problem consisting of finding a sequence of T discrete control inputs $\mathbf{u} = (u_{1,0}, u_{2,0}, ..., u_{1,T-1}, u_{2,T-1})^\top$ and T state vectors $\mathbf{x} = (\mathbf{x}_1, ..., \mathbf{x}_T)^\top$ such that the controls in $\mathbf{u}$ lead to the states in $\mathbf{x}$ under the dynamics in (1), and the final state $\mathbf{x}_T$ is equal to the target state x_f. This formulation is underspecified as more than one sequence of control inputs may lead to the same results, therefore, an additional specification is set in the form of the minimization of a convex function which penalizes the size of the control input. The objective function is then given by

$$f(\mathbf{z}) = \|\mathbf{u}\|_2^2$$

where $f : \mathbb{R}^n \mapsto \mathbb{R}$ and $\mathbf{z} = [\mathbf{x}; \mathbf{u}],\ \mathbf{z} \in \mathbb{R}^n$, and $n = 4T + 2T$. Dynamic feasibility of the states $\mathbf{x}$ is enforced by the first $4T$ rows of the vector

$$h(\mathbf{z}) = \begin{bmatrix} \mathbf{x}_1 - (\mathbf{x}_0 + \dot{\mathbf{x}}_0\Delta t) \\ \vdots \\ \mathbf{x}_T - (\mathbf{x}_{T-1} + \dot{\mathbf{x}}_{T-1}\Delta t) \\ \mathbf{x}_T - \mathbf{x}_f \end{bmatrix} = 0$$

where Δt is the time interval between a pair of states. The last row ensures that the final state matches the target state. The full optimization problem to be solved can then be stated as

$$\min_{\mathbf{z}} \ f(\mathbf{z}) \tag{2a}$$

$$\text{subject to } h(\mathbf{z}) = 0\,, \tag{2b}$$

where $h : \mathbb{R}^n \mapsto \mathbb{R}^m$ and $m = 4(T + 1)$. The problem formulation in (2) does not account for any obstacle-avoiding behavior. However, this can be specified through the use of suitable inequality constraints. This results in the following nonlinear program

$$\min_{\mathbf{z}} \ f(\mathbf{z}) \tag{3a}$$

$$\text{subject to } h_E(\mathbf{z}) = 0 \tag{3b}$$

$$h_I(\mathbf{z}) \geq 0 \tag{3c}$$

where $h_E : \mathbb{R}^n \mapsto \mathbb{R}^{m_E}$ and $h_I : \mathbb{R}^n \mapsto \mathbb{R}^{m_I}$. The constraints $h_I(\mathbf{z})$ are satisfied only if the car is outside of the space taken by the obstacle.

Solving problems Equations (2) and (3) requires finding a point that satisfies their Karush–Kuhn–Tucker (KKT) conditions of optimality. Problem (2) is only subject to equality constraints so its KKT conditions consist only of a vanishing gradient of the Lagrangian and primal feasibility. They can be formulated as a vector in the following manner:

$$F(\mathbf{z}, \lambda) = \begin{bmatrix} \nabla\mathcal{L} \\ h(\mathbf{z}) \end{bmatrix} = \begin{bmatrix} \nabla f(\mathbf{z}) - A(\mathbf{z})^\top\lambda \\ h(x) \end{bmatrix} = 0$$

where the Lagrangian is

$$\mathcal{L}(\mathbf{z}, \lambda) = f(\mathbf{z}) + \lambda^\top h(\mathbf{z}), \quad \lambda \in \mathbb{R}^m$$

and $A(\mathbf{z})$ is the Jacobian of the constraints given by

$$A(\mathbf{z}) = \begin{bmatrix} \nabla h_1(\mathbf{z}) & \dots & \nabla h_m(\mathbf{z}) \end{bmatrix}^\top = 0.$$

A similar vector can be constructed for (3), however, in satisfying the KKT conditions one must also take care to satisfy dual feasibility and complementary slackness. In both cases, solving for an optimal point requires finding the root of a system nonlinear system of equations. To do this, I turned my attention to Newton's method.

IV. Newton's Method for Nonlinear Equations

Newton's method is an algorithm for minimizing an objective function f by iteratively minimizing approximations of f. It starts with an initial guess x_0 and uses Taylor series to form a model function (traditionally quadratic) of f at that point. Then, by minimizing the model function, the algorithm generates a step p_k which is used to compute the new iterate $x_{k+1} = x_k + p_k$ which, under ideal conditions, is closer to the minimizer of f. Newton's method can also be used in the context of nonlinear equations, particularly when the goal is to find a solution for a system of equations of the form

$$r(x) = 0$$

In this case, the model function is generated by taking only the linear terms of the Taylor series of $r(x)$. This avoids potentially having to deal with tensors while still maintaining rapid convergence properties. A linear approximation of $r(x)$ at iterate x_k is given by

$$M_k(p) = r(x_k) + J(x_k)p \tag{4}$$

where $J(\cdot)$ denotes the Jacobian matrix. Then, the Newton step at iterate k is found by solving for p such that $M_k(p) = 0$. Algorithm 1 puts everything together.

Algorithm 1: Newton's Method for Nonlinear Equations

```
  Input: $x_0$
1 for k=0,1,2,... do
2     Solve for $p_k$ in $J(x_k)p_k = -r(x_k)$
3     $x_{k+1} \leftarrow x_k + p_k$
4 return $x_k$
```

Unfortunately, this method is not without its faults. Potential modes of failure are that it may not converge if the initial guess is too far from the solution, it may be too expensive to perform the matrix inversion or calculate explicit derivative information for the nonlinear equations, and depending on the system, $J(x)$ may be singular. However, when special care is taken to address these potential pitfalls, this algorithm provides the foundations for solving much more complex problems, as we will see later on.

V. Sequential Quadratic Programming

Sequential quadratic programming is a method for efficiently solving constrained nonlinear optimization problems. The basic idea behind SQP is to iteratively step from an initial guess x_0 towards the solution x^* by solving a series of quadratic subproblems. Consider an equality-constrained problem with decision variable x and solution x^*:

$$\min_x \quad f(x) \tag{5a}$$

$$\text{subject to } h(x) = 0\,, \tag{5b}$$

where $f : \mathbb{R}^n \mapsto \mathbb{R}$ and $h : \mathbb{R}^n \mapsto \mathbb{R}^m$ are both smooth functions. The KKT conditions are given by

$$F(x,\lambda) = \begin{bmatrix} \nabla \mathcal{L} \\ h(x) \end{bmatrix} = \begin{bmatrix} \nabla f(x) - A(x)^\top \lambda \\ h(x) \end{bmatrix} = 0 \tag{6}$$

where the Lagrangian is

$$\mathcal{L}(x,\lambda) = f(x) + \lambda^\top h(x), \quad \lambda \in \mathbb{R}^m$$

and $A(x)$ is the Jacobian of the constraints given by

$$A(x) = \begin{bmatrix} \nabla h_1(x) & \dots & \nabla h_m(x) \end{bmatrix}^\top = 0.$$

(6) is a system of $n+m$ equations (possibly nonlinear) with $n+m$ unknowns, x and λ. One way of solving for these unknowns is to use Newton's method as described in Section IV. This entails starting at an initial guess/iterate (x_k, λ_k) of the decision variable and the Lagrange multipliers, and then stepping towards the solution using a Newton step $(p_{k,x}, p_{k,\lambda})$

$$\begin{bmatrix} x_{k+1} \\ \lambda_{k+1} \end{bmatrix} = \begin{bmatrix} x_k \\ \lambda_k \end{bmatrix} + \begin{bmatrix} p_{k,x} \\ p_{k,\lambda} \end{bmatrix} \tag{7}$$

where each Newton step is obtained by computing the root of a linear approximation for (6). This root is the solution of the following system of equations:

$$F'(x_k,\lambda_k) \begin{bmatrix} p_{k,x} \\ p_{k,\lambda} \end{bmatrix} = -F(x_k,\lambda_k). \tag{8}$$

Where $F'(x,\lambda)$ is the Jacobian of (6) with respect to (x,λ) and is given by

$$F'(x,\lambda) = \begin{bmatrix} \nabla^2_{xx}\mathcal{L}(x,\lambda) & -A(x)^\top \\ A(x) & 0 \end{bmatrix} = 0 \tag{9}$$

It is important to note that (8) only has a solution if (9) is non-singular which is always true under the following assumptions:

Assumption 1 (LICQ)**.** *The set of active constraint gradients is linearly independent and $A(x)$ has full row rank. This assumption is also known as the linear independence constraint qualification (LICQ) and is a standard way of ensuring that linearization of the constraint gradients in (9) is a good approximation of the true gradients.*

Assumption 2 (Positive-definite Hessian)**.** *The Hessian of the Lagrangian $\nabla^2_{xx}\mathcal{L}(x,\lambda)$ is positive definite on the tangent space of the constraints. That is, $d^\top \nabla^2_{xx}\mathcal{L}(x,\lambda)d > 0 \ \forall d \neq 0$ such that $A(x)d = 0$.*

Under these assumptions, the Newton iteration (7) can be shown to converge quadratically provided that the initial guess is "close enough" to x^*. Then, a simple SQP solver is given in Algorithm 2.

Algorithm 2: Local SQP Algorithm for Equality-Constrained Nonlinear Program

```
  Input: $(x_0, \lambda_0)$
1 k = 0
2 while Convergence criteria not met do
3     Evaluate
        $f(x_k), \nabla f(x_k), \nabla^2_{xx}\mathcal{L}(x_k), h(x_k),$ and $A(x_k)$
4     Solve for Newton step $p_k$ using (8)
5     $\begin{bmatrix} x_{k+1} \\ \lambda_{k+1} \end{bmatrix} \leftarrow \begin{bmatrix} x_k \\ \lambda_k \end{bmatrix} + \begin{bmatrix} p_{k,x} \\ p_{k,\lambda} \end{bmatrix}$
6     $k \leftarrow k + 1$
7 return $(x_k, \lambda_k)$
```

Up to this point, it is unclear where the word "quadratic" in SQP is coming from, as the only approximation being done in Algorithm 2 is, in fact, linear. However, it turns out that solving for the Newton step using (8) is equivalent to solving a quadratic program. To see why, we begin by subtracting

$A(\mathbf{x})^\top \lambda$ from both sides of the first equation in (8) and re-writing the vector equation as

$$\begin{bmatrix} \nabla^2_{xx}\mathcal{L}(x_k,\lambda_k) & -A(x_k)^\top \\ A(x_k) & 0 \end{bmatrix} \begin{bmatrix} p_{k,x} \\ \lambda + p_{k,\lambda} \end{bmatrix} = - \begin{bmatrix} \nabla f(x_k) \\ h(x_k) \end{bmatrix}$$

Letting $l_k = \lambda_k + p_{k,\lambda}$, and dropping the subscripts for $p_{k,x}$ the system of equations becomes

$$\begin{aligned} (\nabla^2_{xx}\mathcal{L}(x_k,\lambda_k)p + \nabla f(x_k)) - A(x_k)^\top l_k = 0 \\ A(x_k)p + h(x_k) = 0. \end{aligned}$$

Which can be interpreted as the KKT conditions of the quadratic problem

$$\min_p \ \frac{1}{2}p^\top \nabla^2_{xx}\mathcal{L}(x_k,\lambda_k)p + \nabla f^\top(x_k)p + f(x_k) \tag{10a}$$

$$\text{subject to } A(x_k)p + h(x_k) = 0 \tag{10b}$$

This is a convex problem as $\nabla^2_{xx}\mathcal{L}(x_k,\lambda_k)$ is positive-definite under Assumption 2, and the constraints define an affine sub-space. The solution is a step p_k that minimizes a quadratic approximation of the Lagrangian $L(x,\lambda)$ at iterate pair (x_x,λ_k) such that the step is also the root of a linear approximation of the constraints $h(x)$ at x_k. The corresponding Lagrange multipliers l_k are, by definition, equal to the next iterate of the Lagrange multipliers of the original problem, namely λ_{k+1}.

This interpretation of the solution to the KKT-Newton system of equations in equation (8) is extremely useful as it allows one to extend SQP for use on nonlinear programs with inequality constraints. Concretely, for the general nonlinear program

$$\min_x \ f(x) \tag{11a}$$

$$\text{subject to } h_E(x) = 0 \tag{11b}$$

$$h_I(x) \geq 0 \tag{11c}$$

where $h_E : \mathbb{R}^n \mapsto \mathbb{R}^{m_E}$ and $h_I : \mathbb{R}^n \mapsto \mathbb{R}^{m_I}$, the quadratic subproblem (12) with minimizer p^* and Lagrange multipliers l^* correspond to Newton step p_k and Lagrange multipliers λ_{k+1}, respectively.

$$\min_p \ \frac{1}{2}p^\top \nabla^2_{xx}\mathcal{L}(x_k,\lambda_k)p + \nabla f^\top(x_k)p + f(x_k) \tag{12a}$$

$$\text{subject to } A_E(x_k)p + h(x_k) = 0 \tag{12b}$$

$$A_I(x_k)p + h(x_k) \geq 0. \tag{12c}$$

A practical framework is summarized in Algorithm 3. This algorithm is more general, but requires a new crucial piece: a solver for a quadratic programs with inequality constraints.

Algorithm 3: Local SQP Algorithm for General Non-linear Program

```
Input: $(x_0, \lambda_0)$
1 k = 0
2 while Convergence criteria not met do
3     Evaluate
      $f(x_k), \nabla f(x_k), \nabla^2_{xx}\mathcal{L}(x_k), h(x_k),$ and $A(x_k)$
4     Solve QP (12) to obtain $(p^*, l^*)$
5     $x_{k+1} \leftarrow x_{k+1} + p^*$
6     $\lambda_{k+1} \leftarrow l^*$
7     $k \leftarrow k + 1$
8 return $(x_k, \lambda_k)$
```

VI. Interior-Point Methods

In order to solve the quadratic subproblem in Algorithm 3, my advisor suggested looking into interior-point methods. These algorithms handle inequality constraints by defining a "barrier" problem in which the original objective function is modified with a logarithmic term which greatly penalizes any solution that is close to activating any of the inequality constraints. This guarantees that the solution is always in the feasible set while avoiding having to explicitly deal with any of the inequality constraints. Concretely, a general nonlinear program (11) can be approximated by the barrier problem

$$\min_{x,s} \ f(x) - \mu \sum_{i=1}^{m_I} \log s_i \tag{13a}$$

$$\text{subject to} \quad h_E(x) = 0 \tag{13b}$$

$$h_I(x) - s = 0 \tag{13c}$$

where $s \in \mathbb{R}^{m_I}$ is known as the slack term, and μ is a positive parameter. As $s \to 0$ the constraints in the problem approach those of the original problem if all the inequality constraints were active. However, the addition of the barrier term in the objective prevents this from happening as $-\log s_i \to \infty$ the closer s is to zero, thus guaranteeing the inequality constraints are always satisfied. The term "interior-point" is derived from the fact that satisfying equations (13c) implies having a solution somewhere in the interior of the space defined by the inequality constraints.

KKT conditions for (13) can be written as follows:

$$\nabla f(x) - A_E^\top(x)y - A_I^\top(x)z = 0, \tag{14a}$$

$$-\mu S^{-1}e + z = 0, \tag{14b}$$

$$h_E(x) = 0, \tag{14c}$$

$$h_I(x) - s = 0. \tag{14d}$$

where $y \in \mathbb{R}^{m_E}$, $z \in \mathbb{R}^{m_I}$, S and Z are diagonal matrices whose diagonal entries are given by the entries of s and z, respectively, and $e = (1,1,...,1)^\top$. Equation (14c) is highly nonlinear as $S \to 0$ which may be problematic if using Newton's method. However, it can be transformed into a quadratic equation by multiplying both sides by S.

Equations (14) are once again a nonlinear system of equations. Therefore, we can apply Newton's method as Section V.

This results in the so-called *primal-dual system*

$$\begin{bmatrix} \nabla^2_{xx}\mathcal{L}(x) & 0 & -A_E^\top(x) & -A_I^\top(x) \\ 0 & Z & 0 & S \\ A_E(x) & 0 & 0 & 0 \\ A_I(x) & -I & 0 & 0 \end{bmatrix} \begin{bmatrix} p_x \\ p_s \\ p_y \\ p_z \end{bmatrix} = - \begin{bmatrix} \nabla f(x) - A_E^\top(x)y - A_I^\top(x)z \\ Sz - \mu e \\ h_E(x) \\ h_I(x) - s \end{bmatrix} \tag{15}$$

which can be used to solve for the Newton steps $p = (p_x, p_s, p_y, p_z)$. In equation (15), the Lagrangian is given by

$$\mathcal{L}(x, s, y, z) = f(x) - y^\top h_E(x) - z^\top (h_I(x) - s).$$

and iteration of the primal and dual variables is given by

$$x_{k+1} = x + \alpha_s^{max} p_x \tag{16a}$$
$$s_{k+1} = s + \alpha_s^{max} p_s \tag{16b}$$
$$y_{k+1} = y + \alpha_z^{max} p_y \tag{16c}$$
$$z_{k+1} = z + \alpha_z^{max} p_z. \tag{16d}$$

Unlike Newton's method in Algorithm 1, this implementation of an interior-point method does not directly add the step onto the previous iterate. Instead, it is first scaled it using the "fraction to the boundary" rule in equation (17), which prevents s and z from approaching zero too quickly.

$$\alpha_s^{max} = \max(\alpha \in (0, 1] : s + \alpha p_s \geq (1 - \tau)s) \tag{17a}$$
$$\alpha_z^{max} = \max(\alpha \in (0, 1] : z + \alpha p_z \geq (1 - \tau)z) \tag{17b}$$

A concrete implementation of an interior-point method is provided in Algorithm 4, where $E(x_k, s_k, y_k; \mu_k)$ denotes an error function given by

$$\begin{aligned} &E(x, s, y; \mu) = \\ &\max(\|\nabla f(x) - A_E^\top(x)y - A_I^\top(x)z\|, \\ &\|Sz - \mu e\|, \|c_E(x)\|, \|c_I(x) - s\|) \end{aligned} \tag{18}$$

and μ_k is the barrier parameter. This parameter is updated according to the Fiacco-McCormick approach where it is held fixed until the KKT conditions in (14) are satisfied to a certain accuracy. Then, it updated using the rule

$$\mu_{k+1} = \sigma_k \mu_k. \tag{19}$$

In this investigation σ_k was chosen to be a constant with a value of $\sigma_k = 0.2$ for the sake of simplicity. However, more sophisticated algorithms commonly apply adaptive changes to σ_k, selecting smaller values as the iterations begin to converge on a solution. The final SQP algorithm is identical to Algorithm 3 but now the quadratic subproblem in line 4 is solved using Algorithm 4.

Algorithm 4: Basic Interior-Point Algorithm

```
  **Input:** $x_0, s_0, y_0, z_0$
1 **while** *Stopping test is not satisfied* **do**
2     **while** $E(x_k, s_k, y_k; \mu_k) \geq \mu_k$ **do**
3         Solve (15) to get search direction $p$
4         Compute $\alpha_s^{max}$ and $\alpha_z^{max}$ using (17)
5         Compute $(x_{k+1}, s_{k+1}, y_{k+1}, z_{k+1})$ using (16)
6         $\mu_{k+1} \leftarrow \mu_k$ and $k \leftarrow k + 1$
7     Select new $\mu_k$ using (19)
8 **return** $x_k, s_k, y_k, z_k$
```

VII. Implementation Details

I began implementation by writing Algorithm 2 in Julia. This ensured that basic SQP was working before attempting to combine it with any additional algorithms. Specific implementation details can be examined in the GitHub repository [12] containing code and examples. However, a couple details are worth being mentioned here. Convergence criteria was defined as being met once the merit function $\phi(\mathbf{z})$ in (20) fell below a certain value ϕ_{min}. Note that this merit function is written in terms of the KKT vector in (6), and driving it to zero is equivalent to finding an optimal point for the problem. It follows that the ideal choice of ϕ_{min} is zero, however, the numerical nature of Algorithm 2 means that very rarely will Algorithm 2 converge to point where $\phi(\mathbf{z})$ is exactly equal to zero. Therefore, ϕ_{min} was manually chosen via trial and error, making a trade off between accuracy of the solution and the computation time. Another notable implementation detail is the use of automatic differentiation in the evaluation of gradients and Hessians (symbolic would have been fine too) as opposed to any numerical differentiation techniques. The fact that the explicit and smooth objective function and constraints were available meant that computation of exact derivatives was possible.

$$\phi(\mathbf{z}) = \left\| \begin{bmatrix} \nabla f(x) - A(x)^\top \lambda \\ h(x) \end{bmatrix} \right\|_2^2 \tag{20}$$

After making sure basic SQP was working, I implemented Algorithm 3 and combined it with Algorithm 4. Again, details can be found in the code repository. The one detail worth mentioning is the use of a simple counter as a stopping test for the outer `while` loop in Algorithm 4. I tuned the number of iterations manually and found anywhere from 15-20 to be sufficient for the demonstrations in the next section.

VIII. Results

The initial demonstration problem **P1** was aimed at testing the capabilities of my implementation of Algorithm 2. **P1** was defined with with parameters $T = 100$, $\Delta t = 0.1$, no obstacles, and the following initial and target states:

$$x_0 = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \qquad x_f = \begin{bmatrix} 0 \\ 5 \\ \pi \\ 0 \end{bmatrix}$$

In this scenario the car starts from rest at the origin and facing to the right. It should then move towards a final state 5 units above the start, also at rest, and facing to the left. Figures 2 and 3 show the resulting trajectories and were generated from a initial guesses $\mathbf{z}_0 = \mathbf{1}$ and $\lambda_0 = \mathbf{1}$, and a final merit function of $\phi_5 = 1.98 \times 10^{-15}$ after only 5 iterations.

The second demonstration problem **P2** was aimed at testing the capabilities of Algorithms 3 and 4. It was almost identical to **P2** except that now an obstacle was added in the form of a wall at $x = 1$, constraining the feasible set to $x \leq 1$. Due to instabilities in my implementation of Algorithm 4, I had to reduce the number of time steps to $T = 7$ with $\Delta t = 1$, resulting in a less smooth trajectory. Nevertheless, the algorithm successfully generated a valid trajectory, with a sharper turn that avoids any $x > 1$. The resulting trajectories are seen in Figures 2 and 4 and had a final merit function of $\phi = 0.049$ after 73 iterations.

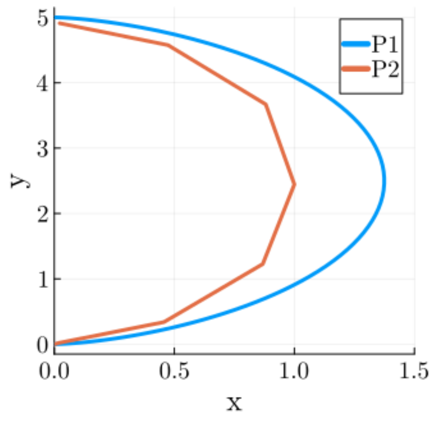

Fig. 2: (x, y) trajectory for **P1** and **P2**

IX. Discussion

Intuitively, a control sequence that solves **P1** and **P2** should accelerate while turning smoothly to the left and then decelerate as it approaches the objective. And indeed both of the generated trajectories do exactly this. However, it is interesting to note how the difference in constraints changed the resulting solution. For example, in **P2** the addition of the wall (and reduction of the overall time) required the car to perform a sharper and faster turn. This is clearly shown by the increased magnitude of the control inputs.

Although both implementations were successful for a set of simple scenarios, there still remains much to be improved upon. For example, it took much longer to compute a solution once a simple inequality constraint was added. The addition of the interior-point algorithm in every iteration of the SQP procedure greatly reduced performance, with specific bottlenecks seen, for example, at the explicit computation of gradients, Jacobians and Hessians at every iteration. This was particularly apparent when my implementation of Algorithm 3 began failing when attempting to solve for any **P2** with more than 7 time steps. Instability in the algorithm was not only due

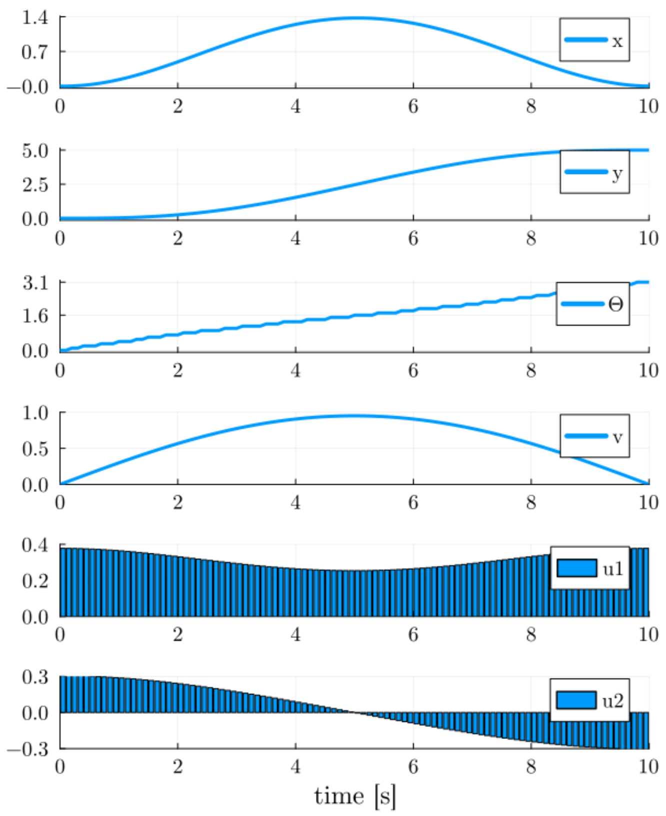

Fig. 3: State and control trajectories for **P1**

to the increased dimensionality of the decision varible, but also because the Newton steps generated by line 4 in Algorithm 3 were too large, causing the iterates to bounce around without ever converging on a local minimum. Adding a manual scaling factor of 0.01 to each step prevented this behavior. However, this was at the great expense of efficiency, as the steps became extremely small and took many more iterations to converge at a solution. The proper solution to this problem is to add a line-search procedure that adaptively reduces the magnitude of the Newton step calculation in order to ensures the resulting iterate has a smaller merit function.

Another source of problematic behavior was the "fraction to the boundary rule" described in (17). In practice, the evaluation of this rule required the generation of discretized $\alpha \in (0, 1]$ which was hard to do in an efficient manner. Too large of a discretization and the algorithm would frequently run into issues when α_s^{max} or α_z^{max} turned out to be empty sets. Too small of a discretization and evaluation of (17) became a significant bottleneck. The implementation phase of this investigation was characterized by the discovery of many of these practical problems which are never apparent when deriving the theory. However, before spending too much time trying to make the interior-point algorithm work better, I recalled the all-important maxim: "*garbage in, garbage out*." The SQP procedure defined in Algorithm 3 is relatively bare-bones, and not equipped with any robustness features. If, and when, issues like inconsistent linearizations of the constraints

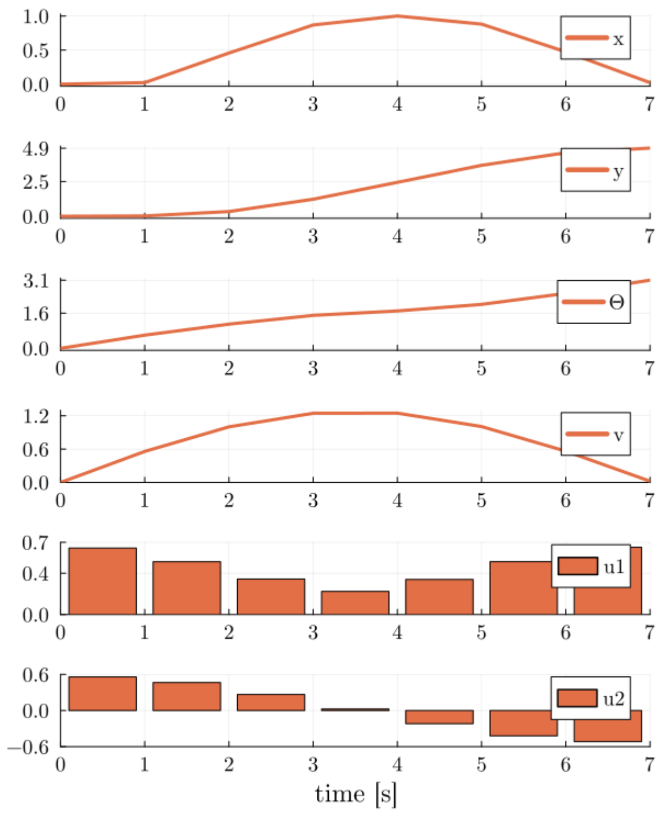

Fig. 4: State and control trajectories for **P2**

arise, or when the Hessian $\nabla^2_{xx}\mathcal{L}$ evaluated at a certain iterate comes close to singularity, the quadratic subproblem fed into the interior-point algorithm will be ill-defined from the very moments of its conception, and will no doubt fail or return incorrect results. Therefore, potential next steps are adding robustness to Algorithm 3, perhaps in the form of objective function reformulations that avoid inconsistencies or quasi-Newton approximations of the Hessian $\nabla^2_{xx}\mathcal{L}$ that are cheaper to evaluate and enforce positive-definiteness. And once that is done, it could be worth doing the same for the interior-point algorithm through the addition of proper line-searching or trust-region procedures, or the development of more sophisticated merit functions.

However, before attempting to optimize or bulletproof any of the algorithms, it might be worth zooming out even more - all the way to the very formulation of the problem at hand. Why define the decision variable in terms of both the control sequence and the trajectory? Perhaps this adds an unnecessary burden onto the algorithm, as it must now iterate in a massive decision space of not just an entire control sequence, but also the resulting states. The dynamics of the system are known with full certainty, and given a control sequence one is easily able to determine the resulting state trajectory. Why then, ask the algorithm to numerically iterate towards this trajectory as well? Problem 1 in Homework 4 states a version of this problem where the decision variables are just the controls and do not include the trajectories. However, the system there has linear dynamics, whereas the one described in this investigation does not. This non-linearity surely adds complexity, but it is still worth investigating whether the problem in this investigation (or any problem for that matter) can be formulated in terms of a decision variable with less dimensions.

X. Conclusion

This project was an immensely fun and fulfilling. Starting with a simple, practical problem I wanted to solve, I dove into the gory details of what it takes to implement a set of proper optimization algorithms. Towards the end of my investigation I was informed of the existence of IPOPT (short for Interior Point Optimizer) [15], a popular optimization library which implements many of the procedures I outlined in a very similar manner. At first, this was a bit disheartening, but then I realized that perhaps the most rewarding part of this investigation was the effort itself. I am grateful for the opportunity to deeply research something I find interesting, and this investigation leaves me excited to learn more about the subject and full of ideas of what to try next.

XI. Acknowledgements

I'd like to thank Dr. Efstathios Bakolas for encouraging us to pursue something of interest as part of our class. I'd also like to thank my advisor Dr. David Fridovich-Keil for his guidance throughout this project.

References

[1] Erling D. Andersen and Knud D. Andersen. *The Mosek Interior Point Optimizer for Linear Programming: An Implementation of the Homogeneous Algorithm*, pages 197–232. Springer US, Boston, MA, 2000. ISBN 978-1-4757-3216-0. doi: 10.1007/978-1-4757-3216-0_8. URL https://doi.org/10.1007/978-1-4757-3216-0_8.

[2] Alexus Barclay, Philip E. Gill, and J. Ben Rosen. Sqp methods and their application to numerical optimal control. 1998.

[3] Richard H. Byrd, Jorge Nocedal, and Richard A. Waltz. Knitro: An integrated package for nonlinear optimization. 2006.

[4] Yuying Chen, Haoyang Ye, and Ming Liu. Hierarchical trajectory planning for autonomous driving in low-speed driving scenarios based on rrt and optimization, 2019. URL https://arxiv.org/abs/1904.02606.

[5] Philip E. Gill, Walter Murray, and Michael A. Saunders. Snopt: An sqp algorithm for large-scale constrained optimization. *SIAM Rev.*, 47:99–131, 2002.

[6] M Glavic and Louis Wehenkel. Interior point methods: A survey, short survey of applications to power systems, and research opportunities. 02 2004.

[7] Jacek Gondzio and Andreas Grothey. Solving non-linear portfolio optimization problems with the primal-dual interior point method. *European Journal of Operational Research*, 181(3):1019–1029, 2007. ISSN 0377-2217. doi: https://doi.org/10.1016/j.ejor.2006.03.

006. URL https://www.sciencedirect.com/science/article/pii/S037722170600138X.

[8] Dylan Hadfield-Menell, Christopher Lin, Rohan Chitnis, Stuart Russell, and Pieter Abbeel. Sequential quadratic programming for task plan optimization. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5040–5047, 2016. doi: 10.1109/IROS.2016.7759740.

[9] T Y Huang and W J Chen. Application of sequential quadratic programming for obtaining optimal obstacle avoidance posture and end position of a 7-axis robotic manipulator. 1113(1):012002, mar 2021. doi: 10.1088/1757-899X/1113/1/012002. URL https://dx.doi.org/10.1088/1757-899X/1113/1/012002.

[10] Steven M. LaValle. Rapidly-exploring random trees : a new tool for path planning. *The annual research report*, 1998.

[11] Jorge Nocedal and Stephen Wright. *Numerical optimization*. Springer Verlag, 2006.

[12] Fernando Palafox Escobedo. Sqp solver for julia. https://github.com/fernandopalafox/sqp-solver.jl, 2022.

[13] Ankit R Patel, Mahesh A Patel, and Dhaval R Vyas. Variational analysis and sequential quadratic programming approach for robotics. *Procedia Technology*, 4:636–640, 2012. ISSN 2212-0173. doi: https://doi.org/10.1016/j.protcy.2012.05.102. URL https://www.sciencedirect.com/science/article/pii/S2212017312003817. 2nd International Conference on Computer, Communication, Control and Information Technology(C3IT-2012) on February 25 - 26, 2012.

[14] Christopher V Rao, Stephen J Wright, and James B Rawlings. Application of interior-point methods to model predictive control. *Journal of optimization theory and applications*, 99(3):723–757, 1998.

[15] Andreas Wächter and Lorenz T. Biegler. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical Programming*, 106:25–57, 2006.