Finding the Limits of Machine Learning in Optimization

David A. Edwards[1]†, Binan Gu[2], Katherine Johnston[3], Maia Wichman[4], and Maxim Zyskin[5]

[1] *University of Delaware, Newark, DE, USA*
[2] *New Jersey Institute of Technology, Newark, NJ, USA*
[3] *University of Washington, Seattle, WA, USA*
[4] *University of Nebraska, Lincoln, NE, USA*
[5] *Oxford University, Oxford, UK*

(*Communicated to* MIIR *on September 7, 2022; in revised form September 26, 2022*)

Study Group: 38th Mathematical Problems in Industry Workshop, Worcester Polytechnic Institute, June 13–17, 2022

Communicated by: Burt Tilley, Worcester Polytechnic Institute

Industrial Partner: DEKA Research and Development Corportation, Manchester, NH

Presenter: Derek Kane

Team Members: Darcy Brunk, San Francisco State University; Derek Drumm, Worcester Polytechnic Institute; David A. Edwards, University of Delaware; Pak-Wing Fok, University of Delaware; Binan Gu, New Jersey Institute of Technology; Katherine Johnston, University of Washington; Darsh Nathawani, University of Buffalo; Michael Smith, Worcester Polytechnic Institute; Burt Tilley, Worcester Polytechnic Institute; Tobias Timofeyev, University of Vermont; Maia Wichman, University of Nebraska, Lincoln; Erli Wind-Andersen, New Jersey Institute of Technology; Zhongqiang Zhang, Worcester Polytechnic Institute; Maxim Zyskin, University of Oxford.

Industrial Sector: Computing/Robotics

Key Words: Robotics; machine learning; gradient optimization; variational principles; supervised learning.

MSC2020 Codes: 70B15, 70E60, 68T07, 93B47, 93C85

† Corresponding Author: `dedwards@udel.edu`

Summary

The initial position and velocity of a robot is given, and the problem posed is to make it stop at the origin in the shortest possible time, given a maximum acceleration and speed. The robot can control its acceleration vector, and hence the full optimization problem can be specified as a Hamiltonian system where the solution will minimize the transit time. This problem is discussed in both the one- and two-dimensional cases. The key control parameter is the acceleration direction; reducing the problem to a one-dimensional optimization opens up several areas of exploration. The direction can be optimized using a global search algorithm, or can be updated periodically using a local search algorithm with a penalty function. Numerical solutions are presented in these cases, including when physical obstacles are included in the penalty function. The one-dimensional optimization also allows the use of reinforced learning to minimize the transit time.

1 Introduction

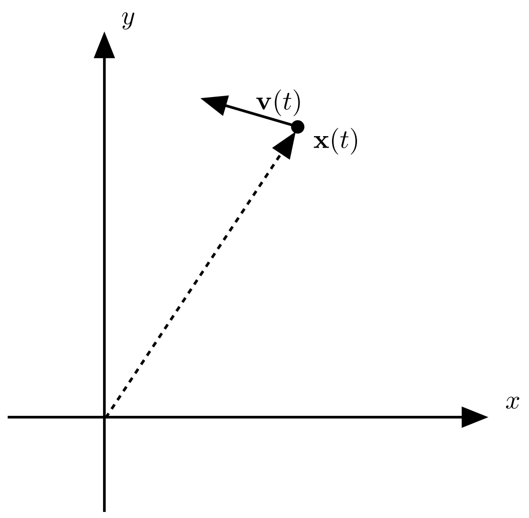


Figure 1.1. Schematic of robot movement.

We consider the problem of a robot (position $\mathbf{x}(t)$) which moves with velocity $\mathbf{v}(t)$ (see Fig. 1.1). The robot is given a simple problem: move to the origin and stop there.

We assume that:

(1) initially the robot has position $\mathbf{x}_0$ and velocity $\mathbf{v}_0$.

(2) the robot can accelerate instantaneously in any direction with a maximum acceleration of $a_{\max}$.

(3) the robot knows its position and velocity at any time.

(4) the only instructions the robot can execute are a change in **a**, the acceleration vector.

The final consideration is the treatment of the speed, which is assumed to have a maximum value. In reality, this is because its maximum acceleration is not enough to overcome frictional forces at high speed. There are surprisingly many models for this, though one useful one is [3, p. 60]

$$\dot{\mathbf{v}} = \mathbf{a} - \nu \mathbf{v}, \tag{1.1}$$

which has a steady state of $v_{\max} = a_{\max}/\nu$, as desired.

Treating the more physically realistic model (1.1) is beyond the scope of this manuscript. Instead, for simplicity we use the standard frictionless model $\dot{\mathbf{v}} = \mathbf{a}$ coupled to an imposed maximum speed constraint $|\mathbf{v}| = v_{\max}$. Though easy enough to implement in software, this form of the constraint wreaks havoc on analytical solutions.

This problem can be attacked various ways, as illustrated in this report. In Sec. 2, we outline the governing equations for the problem. In Sec. 3, we implement a series of numerical algorithms to guide the robot in the case of no obstacles, and generalize to the case of obstacles in Sec. 4. In Sec. 5 we introduce machine and supervised learning approaches, and in Sec. 6 we present some analytical solutions to the problem.

2 Governing Equations

2.1 The One-Dimensional Problem

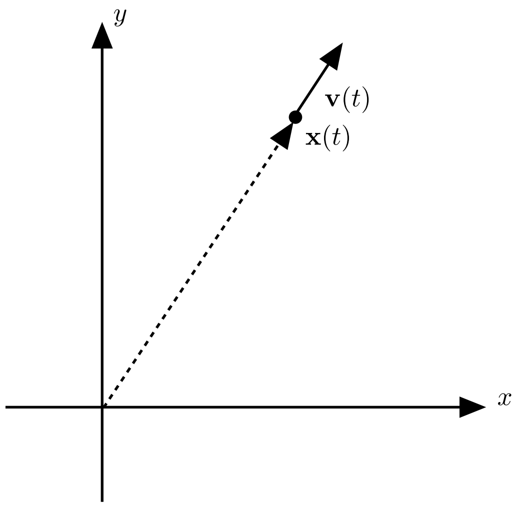


Figure 2.1. Schematic of aligned robot movement.

We consider what at first might seem a very specialized case: one where the velocity of the robot is aligned with its position vector (see Fig. 2.1). In this case, the system reduces to a one-dimensional problem in the radius $r = |\mathbf{x}|$. However, it will be shown in

Sec. 3.2 that even when starting with an arbitrary velocity, the robot will spend quite a bit of time in this "one-dimensional" state.

We begin by considering the case where the robot is "some distance" from the origin (this phrase will be formally defined later). Then the robot should apply maximum acceleration towards the origin:

$$\ddot{r}(t) = -a_{\max}. \tag{2.1}$$

We assume that the acceleration continues until the robot reaches a speed of $v_{\max}$. Motivated by the physics associated with friction, we further assume that the "acceleration" (really force) must be maintained at the maximum level to keep the speed at $v_{\max}$.

When the robot nears the origin, it must begin to slow to achieve its goal of reaching the origin with zero velocity. Again, for the shortest elapsed time, the deceleration must be maximized. Since the acceleration is constant, we have that

$$r(t) = r_{\mathrm{d}} + \frac{a_{\max}\tau^2}{2} - v_{\max}\tau, \tag{2.2a}$$

where τ measures the time of deceleration, and r_{d} is the position at which the deceleration begins (which is what we wish to determine). Moreover, examining the velocity, we obtain the following:

$$-v_{\max} + \tau_{\mathrm{d}} a_{\max} = 0 \qquad \Longrightarrow \qquad \tau_{\mathrm{d}} = \frac{v_{\max}}{a_{\max}}, \tag{2.2b}$$

where τ_{d} is the duration of deceleration. Given this definition, at $\tau = \tau_{\mathrm{d}}$ the robot must be at the origin, so (2.2a) becomes

$$\begin{aligned} r_{\mathrm{d}} + \frac{v_{\max}^2}{2a_{\max}} - \frac{v_{\max}^2}{a_{\max}} &= 0 \\ r_{\mathrm{d}} &= \frac{v_{\max}^2}{2a_{\max}}, \end{aligned} \tag{2.3}$$

where we have used (2.2b). This is the standard braking-distance formula from physics [4, p. 64].

What if the robot starts close enough to the origin that it cannot reach maximum speed before having to decelerate? In that case, the robot must begin decelerating whenever it reaches the braking distance for its actual velocity:

$$r_{\mathrm{d}} = \frac{v^2(t)}{2a_{\max}}. \tag{2.4}$$

Equation (2.4) is generic enough to cover both cases, so in the one-dimensional case we have the following algorithm:

Algorithm 1.

Apply maximum acceleration/force until (2.4) *is satisfied, then apply maximum deceleration.*

(Note that this algorithm holds even in the case where we reach maximum velocity, since in that case we still apply maximum force to maintain it.)

2.2 Polar Coordinates

We return to the more general case shown in Fig. 1.1. Given the focus on reducing r, it is convenient to write the relevant equations in terms of polar coordinates:

$$\mathbf{x} = (r\cos\theta, r\sin\theta) = r\hat{\mathbf{r}}, \qquad \hat{\mathbf{r}} = (\cos\theta, \sin\theta), \tag{2.5a}$$

where the hat indicates a unit vector (see Fig. 2.2). $\hat{\theta}$ is perpendicular to $\hat{\mathbf{r}}$:

$$\hat{\theta} = (-\sin\theta, \cos\theta). \tag{2.5b}$$

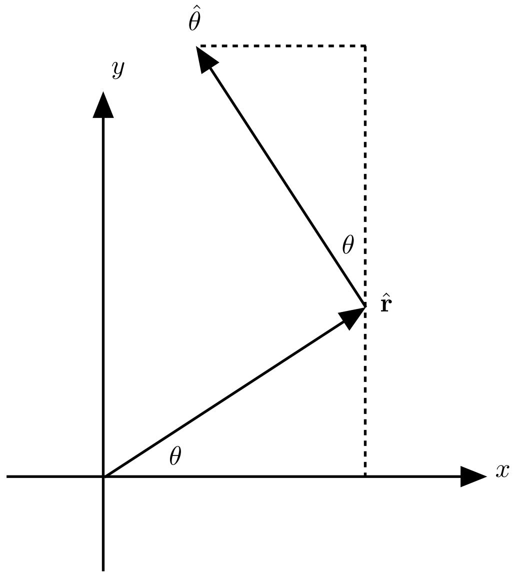


Figure 2.2. Unit vectors, polar coordinates.

To calculate velocities, it is necessary to have the derivative of the normal vectors:

$$\frac{d\hat{\mathbf{r}}}{dt} = (-\dot{\theta}\sin\theta, \dot{\theta}\cos\theta) = \dot{\theta}\hat{\theta}, \tag{2.6a}$$

$$\frac{d\hat{\theta}}{dt} = (-\dot{\theta}\cos\theta, -\dot{\theta}\sin\theta) = -\dot{\theta}\hat{\mathbf{r}}. \tag{2.6b}$$

Then using (2.6), we have

$$\mathbf{v} = \frac{d\mathbf{x}}{dt} = \dot{r}\hat{\mathbf{r}} + r\frac{d\hat{\mathbf{r}}}{dt} = \dot{r}\hat{\mathbf{r}} + r\dot{\theta}\hat{\theta}, \tag{2.7a}$$

$$\begin{aligned}\mathbf{a} = \frac{d\mathbf{v}}{dt} &= \ddot{r}\hat{\mathbf{r}} + \dot{r}\frac{d\hat{\mathbf{r}}}{dt} + \dot{r}\dot{\theta}\hat{\theta} + r\ddot{\theta}\hat{\theta} + r\dot{\theta}\frac{d\hat{\theta}}{dt} \\ &= \ddot{r}\hat{\mathbf{r}} + \dot{r}\dot{\theta}\hat{\theta} + \dot{r}\dot{\theta}\hat{\theta} + r\ddot{\theta}\hat{\theta} - r\dot{\theta}^2\hat{\mathbf{r}} \\ &= (\ddot{r} - r\dot{\theta}^2)\hat{\mathbf{r}} + (r\ddot{\theta} + 2\dot{r}\dot{\theta})\hat{\theta}, \end{aligned} \tag{2.7b}$$

which is a standard result from physics [3, p. 32].

As a first naïve approach, we may reuse our algorithm from the one-dimensional case and take $\mathbf{a}$ to be proportional to $\hat{\mathbf{r}}$ only, in which case the $\hat{\theta}$ component of (2.7b) becomes

$$\begin{aligned} r\ddot{\theta} + 2\dot{r}\dot{\theta} &= 0 \\ \ddot{\theta} &= -\frac{2\dot{r}\dot{\theta}}{r}. \end{aligned} \tag{2.8}$$

But (2.8) illustrates the key difference between the one- and two-dimensional cases. In the one-dimensional case, $\dot{\theta}$ is always zero, so this equation is satisfied exactly. However, in the two-dimensional case, for *any* acceleration imposed only in the radial direction, the angular velocity $\dot{\theta}$ will become arbitrarily large as the robot approaches the origin. (Note that the r term multiplying $\dot{\theta}$ in (2.7a) ensures that a large angular velocity does not violate the bound that $v < v_{\max}$.)

3 Numerical Results, No Obstacles

To validate this analysis, we wrote a Python code. The robot was given an initial condition $\mathbf{x}_0$, where each component was uniformly distributed in a range $[-x_{\max}, x_{\max}]$ (for parameter values, see Appendix A). Each component of $\mathbf{v}_0$ was uniformly distributed in a range $[-v_{\max}, v_{\max}]$, and then $\mathbf{v}_0$ was scaled to have length $v_{\max}$ if the sampled speed was too large.

3.1 Radial Acceleration

The robot's acceleration is given by $\mathbf{a} = -a_{\max}\hat{\mathbf{r}}$. (Because there is now a component of the velocity in the angular direction, we do not impose deceleration once we cross the threshold (2.4).) We employ a simple explicit Euler method with time step dt, keeping in mind the artificial speed constraint that $v \le v_{\max}$. Hence if adding $\mathbf{a}(t)dt$ to the velocity caused the speed to exceed $v_{\max}$, the velocity was scaled as follows:

$$\mathbf{v}(t+dt) = v_{\max}\frac{\mathbf{v}(t) + \mathbf{a}(t)dt}{|\mathbf{v}(t) + \mathbf{a}(t)dt|}$$

to enforce $|\mathbf{v}| = v_{\max}$.

The algorithm ran until two conditions were met:

(1) $r < r_{\mathrm{s}}$, so the robot was within some bound r_{s} of the origin, and

(2) $v < v_{\mathrm{s}}$, so the speed was less than some bound v_{s}.

The bound on the speed is much tighter the one on the radius; the exact values are listed in Appendix A. If the code did not converge within $N_{\max}$ iterations, the algorithm is said to fail.

For a typical run, the robot's path is shown at left in Fig. 3.1. The initial point is the blue star at lower left, and the initial velocity is indicated by the line emanating from it. Each iterate is indicated with a point. Though the robot does approach the origin (black star at upper right), it overshoots it.

Moreover, the algorithm actually does not converge because the velocity does not decrease to zero, as shown at right. Instead the velocity rapidly oscillates when the robot is near the origin. In particular, as the robot approaches the origin, the speed decreases.

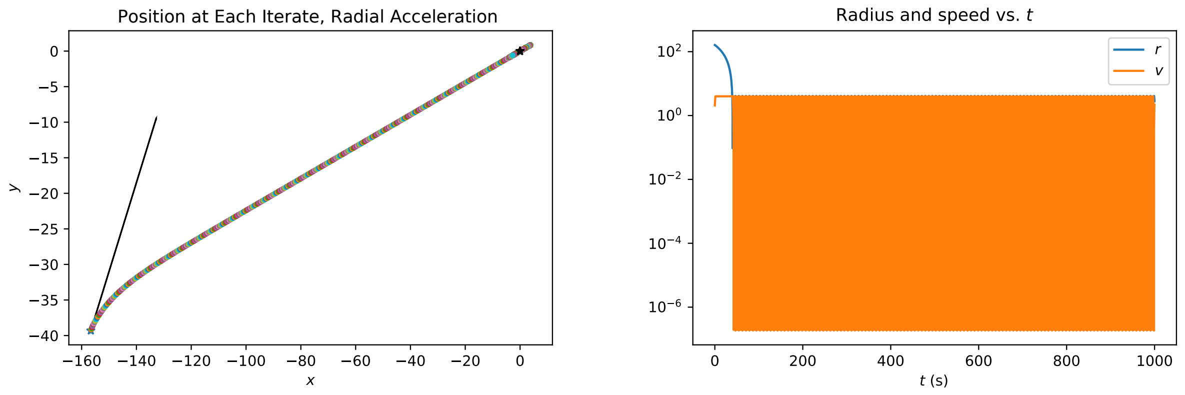


Figure 3.1. Left: Robot path with radial acceleration only. Right: Comparing r and v for various iterates. Here the orange block corresponds to rapid oscillations in v between the upper and lower values.

But as discussed above, with only a radial acceleration, as $r \to 0$, $\dot{\theta}$ gets large, so the robot will near the origin, but then pass beyond it. Hence to reverse this escape trajectory, the speed has to increase again. This process repeats until the process is terminated.

The overshoot phenomenon may be exacerbated by maintaining a fixed time step throughout the experiment. If instead the time step were reduced as the robot approaches the origin, it would no longer stay on the same path for so long, reducing the possibility of overshoot. Moreover, the robot would reevaluate its acceleration direction more frequently, allowing it to keep on a path better aligned with the origin. We did not have time to explore these ideas at the workshop.

3.2 Convex Combination Approach

Since using a purely radial acceleration does not yield convergence, we expand the universe of acceptable accelerations. First consider that if the acceleration is constant, then the movement of the robot is given by

$$\mathbf{x}(t) = \frac{\mathbf{a}t^2}{2} + \mathbf{v}_0 t + \mathbf{x}_0.$$

Therefore, if we want $\mathbf{x}(t) = \mathbf{0}$ at some time, the needed (constant) acceleration would depend on both the initial velocity and the initial position.

Motivated by this fact, in the time-dependent case we try a form for the acceleration which depends on the robot's current velocity and position:

$$\mathbf{a} = -a_{\max}\hat{\mathbf{a}}, \qquad \hat{\mathbf{a}} = \frac{(1-\lambda)\mathbf{v} + \lambda\mathbf{x}}{|(1-\lambda)\mathbf{v} + \lambda\mathbf{x}|}, \quad \lambda \in [0,1]. \tag{3.1}$$

Note the magnitude of the acceleration remains at its maximum, and here $\hat{\mathbf{a}}$ has been defined as the unit vector pointing *opposite* to the acceleration so the first equation in (3.1) is similar to (2.1). The parameter λ determines how much weight we give to opposing the velocity (the first term) versus driving the robot to the origin (the second term).

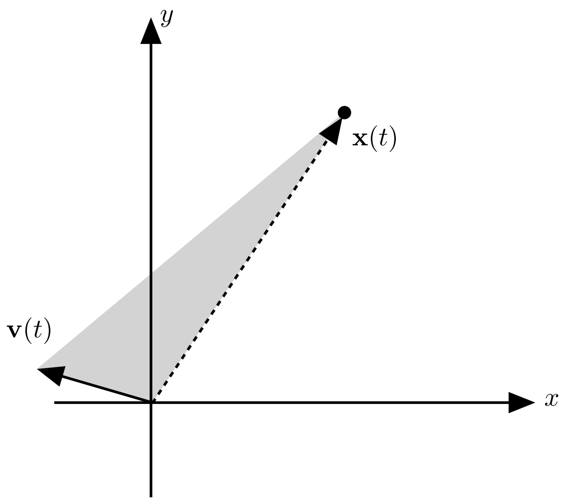


Figure 3.2. Shaded region: allowable $\hat{\mathbf{a}}$ directions with $\lambda \in [0, 1]$.

Mathematically, by taking $\lambda \in [0, 1]$, we are forcing $\hat{\mathbf{a}}$ to be a convex combination of $\mathbf{x}$ and $\mathbf{v}$. Physically, $\hat{\mathbf{a}}$ must lie in a direction between $\mathbf{v}$ and $\mathbf{x}$ (see Fig. 3.2). That choice will come back to haunt us later.

The approach in Sec. 3.1 is equivalent to taking $\lambda = 1$ in (3.1). From (2.8) we know that problems with that approach occur when r gets small, as indicated at the right of Fig. 3.1. For small r, it becomes more important to reduce the speed than to reduce r. But this is exactly what (3.1) prescribes: as r gets small, for any $\lambda \neq 1$ the component of $\hat{\mathbf{a}}$ reducing speed will dominate the component reducing distance.

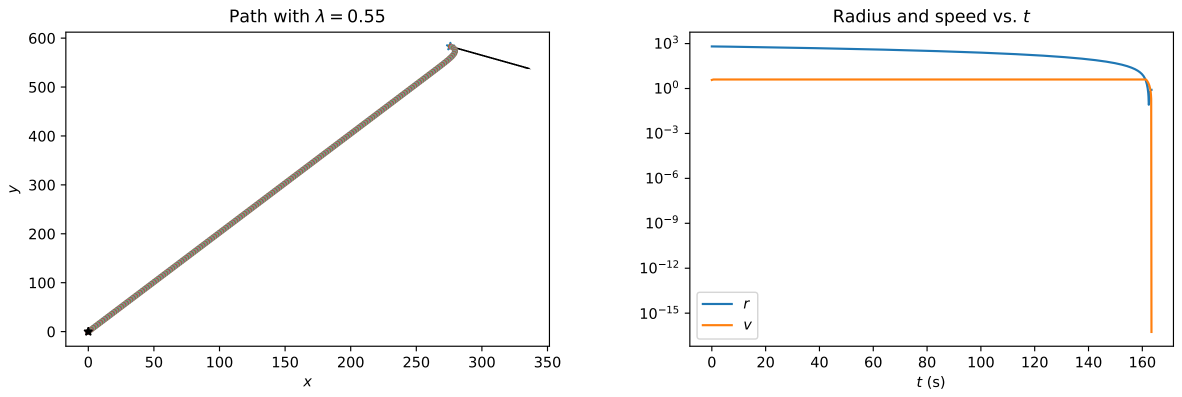


Figure 3.3. Left: Robot path using (3.1) with $\lambda = 0.55$. Right: Comparing the sizes of r and v for various times.

This is illustrated in Fig. 3.3, which uses the same design as Fig. 3.1. The algorithm guides the robot to the origin, as both the radius and speed go to zero. Moreover, the goals are achieved in two stages. First, the robot proceeds at maximum velocity to the origin, roughly in the first 160 s. Then once r gets small, the speed quickly reduces to zero. This illustrates the “switching” nature of (3.1) for moderate and small r.

Once the initial transients decay away, the robot’s trajectory in Fig. 3.3 is nearly radial. But in that case $\mathbf{x}$ and $\mathbf{v}$ are almost parallel, and the shaded region in Fig. 3.2 collapses. Hence along any near-radial path, the restriction on λ in (3.1) is essentially meaningless, as there is not really a one-parameter family of $\hat{\mathbf{a}}$ at all.

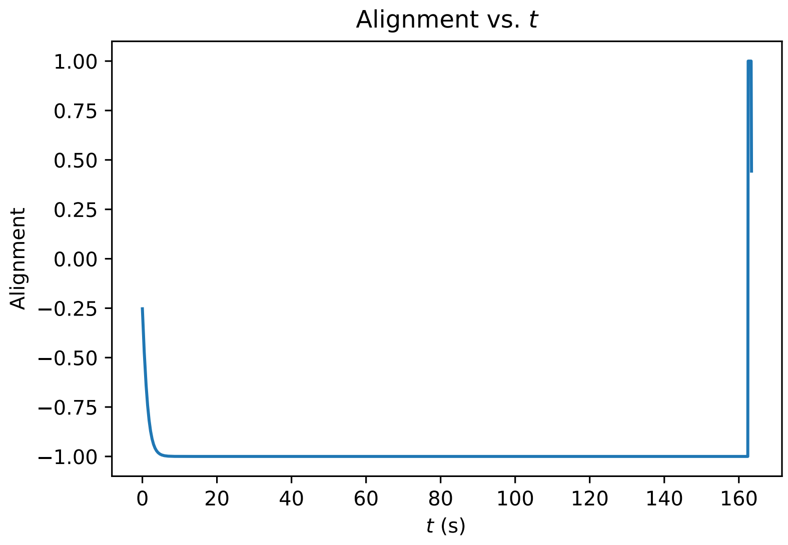


Figure 3.4. Alignment of $\mathbf{x}$ and $\mathbf{v}$ vectors.

In order to quantify this shortcoming, we compute the scalar projection

$$\alpha = \frac{\mathbf{x} \cdot \mathbf{v}}{|\mathbf{x}||\mathbf{v}|}, \tag{3.2}$$

which we call the *alignment*. In Fig. 3.4 we show α *vs.* t for the path shown in Fig. 3.3. Initially the vectors are not perfectly aligned due to the random assignment of $\mathbf{v}_0$ and $\mathbf{x}_0$. Then for most of the run, $\alpha \approx -1$, indicating that $\mathbf{v} \approx -\mathbf{x}$, which corresponds to a radial trajectory. Then near the end of the run, α quickly switches to $+1$, so $\mathbf{v} \approx \mathbf{x}$, which corresponds to the robot overshooting the origin. The vectors remain in nearly that state until the acceleration slows the robot down.

Given that this algorithm achieves the robot’s goal, we must refine it in order to minimize the travel time. Therefore, we investigate the time it takes to get to the origin for various λ, which we interpret as a control parameter. With $\lambda = 1$, the algorithm does not converge before the code ends, as discussed above. At the other extreme ($\lambda = 0$), the code does not converge, either. In that case, the sole function of the acceleration is to reduce the velocity. Hence the radius does not change that much (see Fig. 3.5). Therefore, we expect that there is some $\lambda_* \in (0, 1)$ that minimizes the time to the origin.

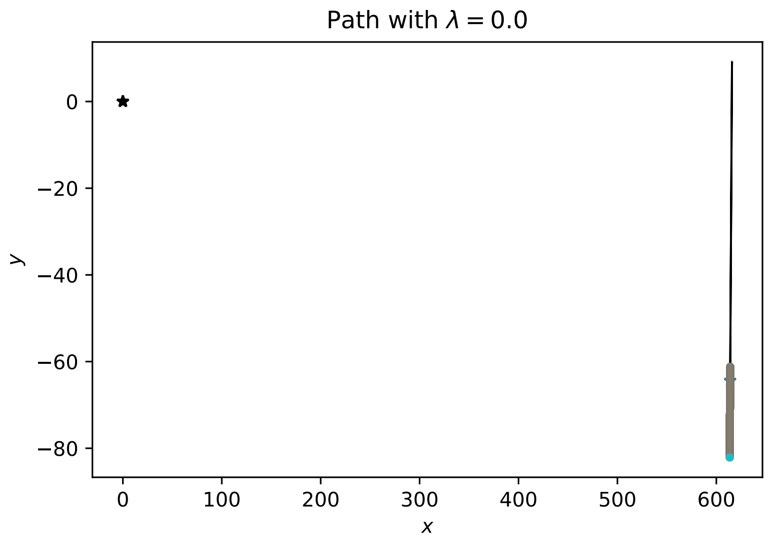


Figure 3.5. Result using (3.1) with $\lambda = 0$.

With a parameter λ that the algorithm can vary, we must then address the following questions:

(1) What set of points should be used for λ? A continuous interval, or a discrete set?

(2) How should values of λ be sampled? Using some probability distribution from the interval/set? With the same sampling, or should the sampling depend on the current position and velocity of the robot?

(3) How should the effectiveness of the algorithm be measured?

(4) How often should the optimization occur?

In Secs. 3 and 4, we use a set of either ten or twenty different values for λ, ignoring the pathological cases 0 and 1. We sample all of the values of λ, rather than taking some smaller set at random. To compare performance among various initial conditions, we calculate the *performance factor* p as follows:

$$p = \frac{N dt}{r_0/v_{\max}} = \frac{\text{robot transit time}}{\text{minimal transit time if } \alpha = 1}, \tag{3.3}$$

where N is the number of time steps to completion. Thus lower values of p are better, and p has a lower bound of 1.

We consider two possible alternatives for question #4, as outlined in the next two subsections.

3.3 Global Search

In the first approach, given some initial state, we simulate the trajectories for various values of λ, which we require to be constant for the entire run. We compare the performance factors of the various λ to find the optimal value λ_*. We then instruct the robot to use that value in (3.1) for the entire run. We refer to this as the *global search* method.

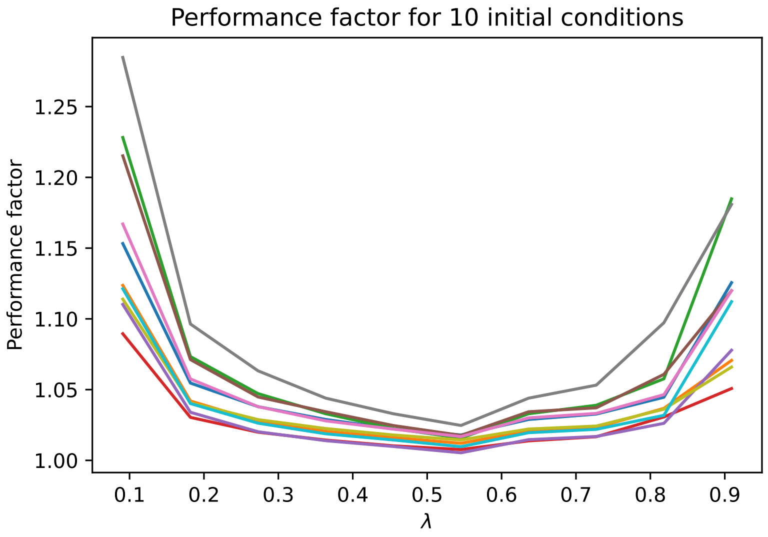


Figure 3.6. Comparing performance factors for different values of λ using (3.1).

Figure 3.6 show the results of this approach for ten different sets of initial conditions, each illustrated by a different color. In the worst cases, the transit time is only about 30% more than the optimal value, while for the best λ values, the transit time is only a few percent above optimal. (This is consistent with the right of Fig. 3.3, which shows $v = v_{\max}$ for the vast majority of the transit time.) Moreover, the graph suggests a single value of λ_* may be near-optimal for any initial state of the robot.

Most of the robot's trajectory is nearly radial (and hence insensitive to λ). Hence differing λ affects only the transient decay at the beginning. The relative brevity of the decay causes variations in p with λ to be minimal, and thus the performance factor in Fig. 3.6 is relatively constant for moderate λ.

This code gives a method by which an optimal parameter $\lambda_*(\mathbf{x}_0, \mathbf{v}_0)$ can be determined for any initial configuration. However, the typical robot does not have the onboard computational power to execute the code efficiently. Therefore, to improve performance, as a preprocessing step we could construct a lookup table of λ_* for a series of initial conditions $(\mathbf{x}_0, \mathbf{v}_0)$. Then we could proceed in one of two different ways:

(1) Given $(\mathbf{x}_0, \mathbf{v}_0)$, use linear interpolation among the entries in the lookup table (`scipy.interpolate.griddata`) to compute λ_*, then use that value for the entire run to get to the origin.

(2) Use the process in #1 to determine the acceleration for some time interval $[0, t_*]$, then repeat the lookup process to find λ_* for the "new starting point" $(\mathbf{x}(t_*), \mathbf{v}(t_*))$. Repeat as needed until the origin is reached.

The lookup table approach has the following drawbacks:

- The table would quickly become lengthy. Entries for every 50 m in $\mathbf{x}$ and fifty points in $\mathbf{v}$ space would produce $21 \times 21 \times 50 = 22{,}050$ entries in the lookup table. However, terabytes of storage could be devoted to this purpose since the robot is performing only an interpolation, so we should be able to get much finer coverage in the lookup table.
- Given the shape of the curves in Fig. 3.6, the entries in the lookup table for λ_* may differ only slightly (or not at all, depending on how many values you sampled). So the computational effort may not be worth it if a single value ($\lambda = 0.55$, say) would provide reasonable results for any initial condition.
- This approach would be best only for the simplest problem we were given: an empty room with no obstacles. Any change in the configuration of the room would require computation of a new table.

3.4 Local Search

Rather than performing a large preprocessing step in order to create a lookup table, it may be more efficient to have the robot perform multiple optimizations on its travels, making course corrections based upon the results. Though these computations would necessarily have to be of smaller size, this procedure has the advantage that it would be able to accommodate changes in the features of the room (obstacles, etc.—see Sec. 4). We outline a very general optimization procedure:

(1) Construct a function $F(\mathbf{x}(t), \mathbf{v}(t); \lambda)$ to be optimized.

(2) At some time t, calculate F for various values of λ, and determine the optimal value λ_*.

(3) For some interval $[t, t + t_*]$, accelerate the robot according to (3.1) using λ_*.

(4) Set $t = t + t_*$ and repeat steps #2 and #3 until the robot reaches the origin.

Some discussion of step (3) is appropriate. In order to minimize computational cost, we would like t_* to be larger than dt.

Since t_* is less than the entire transit time, the optimization outlined above should be quick. (We were not able to do a thorough comparison of runtimes at the workshop.)

The algorithm described in this section is slightly different from the algorithm previously tried by DEKA, where the robot accumulated reward over each time step. (We will return to such an accumulated reward approach in Sec. 5.) In the current formulation, the function is optimized at each interval, and previous results are discarded.

As a first choice for F, we take distance to the origin after the interval in which the acceleration is applied, so we minimize

$$F(\mathbf{x}(t); \lambda) = r(t + t_*). \tag{3.4}$$

Starting with an arbitrary $(\mathbf{x}_0, \mathbf{v}_0)$, we perform the algorithm described above, comparing ten values of $\lambda \in (0, 1)$. We take $t_* = 10$ s, $dt = 1$ s, using the parameters in the Appendix. For reasons that will become clear later, we do not require that F be monotone decreasing from iteration to iteration—we simply choose λ_* to minimize F on the interval given in (3.4).

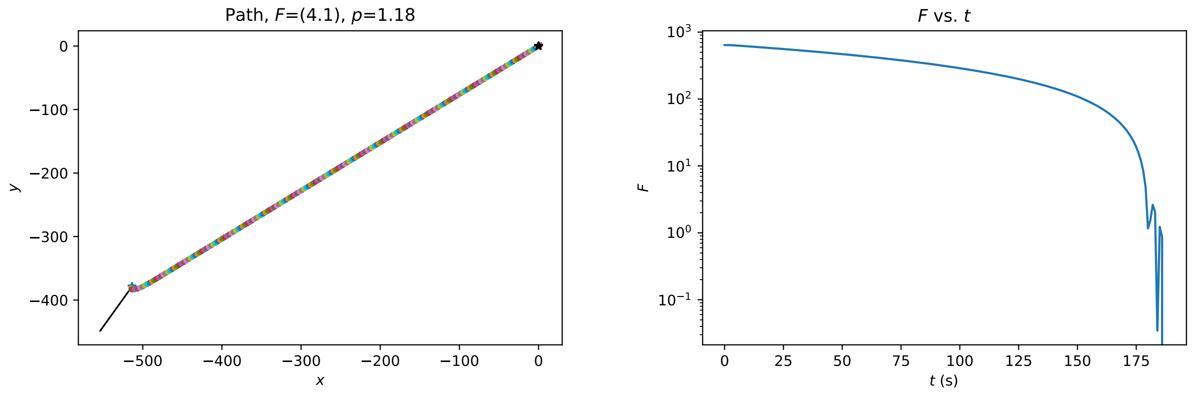


Figure 3.7. Left: Robot path using (3.4). Right: Value of objective function.

A typical trajectory resulting from using (3.4) is shown at left in Fig. 3.7. Though the trajectory converges to the origin, the performance factor (listed in the figure title) is worse than that for the global search. This is consistent with our intuition that solving the global problem will yield a lower minimum time than solving a series of local problems.

At right of Fig. 3.7 we show the evolution of F. Though not clear from the semilog plot, F decays linearly for the bulk of the evolution as the robot takes the radial path. The rise in F at the right of the graph corresponds to the robot overshooting the origin, as seen in the previous section. The robot then decelerates back to the origin with velocity closer to zero.

Since (3.4) does not depend on the velocity, the resulting accelerations would be similar to using (3.1) with $\lambda = 1$, which we know does a poor job stopping the robot. To reduce the overshoot phenomenon, we should penalize high velocities near the origin to force the robot to stop more quickly. Hence we replace (3.4) with the following piecewise function:

$$F(\mathbf{x}(t), \mathbf{v}(t); \lambda) = \begin{cases} r(t + t_*), & r > r_\mathrm{s}, \\ v(t + t_*), & r < r_\mathrm{s}. \end{cases} \tag{3.5}$$

In other words, we reduce the radius until we have satisfied the r_s bound, then switch to minimizing the velocity instead. This sequential process was achieved organically in the original algorithm, as shown at the right of Fig. 3.3. The results from using (3.5) are shown at left in Fig. 3.8. Here the performance factor is somewhat improved from Fig. 3.7.

It may seem artificial to switch from minimizing the radius to minimizing the velocity. One can include both terms, but at some point the dependence of F on v must switch from encouraging high speed to discouraging it. A very simple possibility is listed below:

$$F(\mathbf{x}(t), \mathbf{v}(t); \lambda) = \begin{cases} r(t + t_*) + [v(t + t_*)]^{-1}, & r > r_\mathrm{s}, \\ r(t + t_*) + v(t + t_*), & r < r_\mathrm{s}. \end{cases} \tag{3.6}$$

In other words, when the robot is outside the stopping circle, we want the speed to be large. When the robot is inside the stopping circle, we want the speed to be small. (Obviously the coefficients and exponents of these terms can be varied to optimize results.)

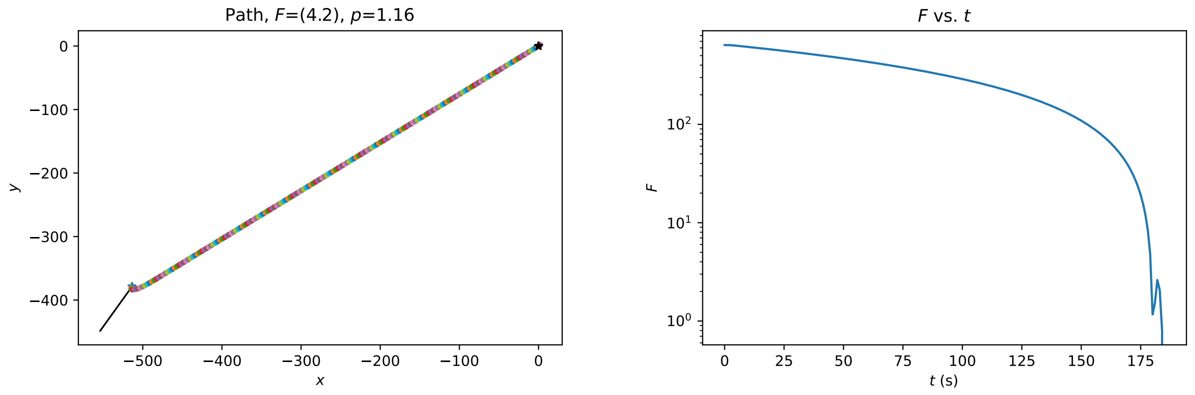


Figure 3.8. Left: Robot path using (3.5). Right: Value of objective function.

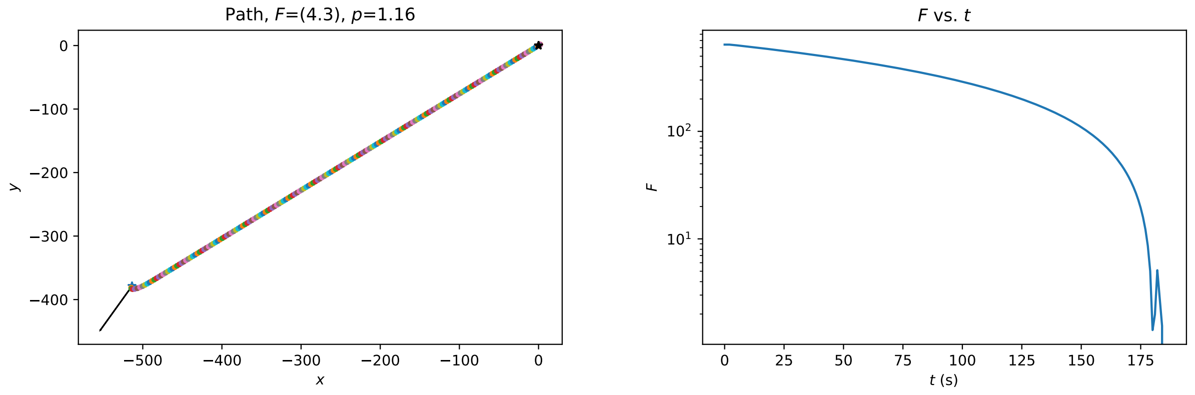


Figure 3.9. Left: Robot path using (3.6). Right: Value of objective function.

The results from using (3.6) are shown in Fig. 3.9, which is virtually indistinguishable from Fig. 3.8. This is because outside the stopping circle, the speed is typically always at $v_{\max}$ anyway, so the v^{-1} term in (3.6) does not affect anything. And inside the stopping circle, the r term is typically smaller than the v term, so there is little difference between (3.6) and (3.5) in that case, either.

4 Including Obstacles

The great advantage of the local search outlined in Sec. 3.4 is that it can adjust to topographical changes on the fly. For instance, if there is some sort of wall or other obstacle between the initial condition and the origin, terms can be added to F to penalize colliding with them. Waypoints are often manually created to help the robot navigate around such obstacles, but including the effect of obstacles into F should reduce the need to do so. This both saves time and makes the robot's behavior more robust.

As a particular case, consider the case of a wall occupying $(x, 200)$, $x > -100$ (see Fig. 4.1). We examine a wall at first rather than a post, chair, or similar object for two reasons. First, it poses a more difficult challenge to the robot, since the robot can avoid it by moving in only one direction. Second, the objective function is somewhat nonstandard.

In particular, we wish to penalize the robot whenever $y \to -200$, as long as $x > -100$. (Otherwise, the robot is the region beyond the wall and can traverse $y = -200$ without difficulty.) The penalty should increase as y gets closer to -200, and become infinite if the robot actually strikes the wall. Hence we posit a penalty term of the form

$$\left[\frac{(x+100)^+}{|y+200|}\right]^\beta, \qquad (x+100)^+ = \max\{0, x+100\}, \qquad \beta > 0. \tag{4.1}$$

As the more complicated cost function (3.6) did not improve results, we use the piecewise form in (3.5) augmented by (4.1):

$$F(\mathbf{x}(t), \mathbf{v}(t); \lambda) = \left[\frac{(x+100)^+}{|y+200|}\right]^\beta + \begin{cases} r(t+t_*), & r > r_\mathrm{s}, \\ v(t+t_*), & r < r_\mathrm{s}. \end{cases} \tag{4.2}$$

Note that for $x < -100$, the barrier term vanishes and we are just left with (3.6), which will then direct the robot to the origin along a radial path.

Using the standard approach outlined in Sec. 3.2 did not work. By the time the robot reached the wall, its velocity and displacement were aligned, so using the convex combination (3.1) provided no way for the robot to divert from the radial path and avoid the barrier.

Therefore, when the vectors are in alignment, we introduce a kludge (which we will systematize in the next section). When $|\alpha| > 0.9$, we replace the vector $\hat{\mathbf{a}}$ in (3.1) with

$$\hat{\mathbf{a}} = \frac{(1-\lambda)\mathbf{z} + \lambda\mathbf{x}}{|(1-\lambda)\mathbf{z} + \lambda\mathbf{x}|}, \qquad \mathbf{z}^T\mathbf{v} = 0. \tag{4.3}$$

With $\mathbf{z}$ replacing $\mathbf{v}$, there are now two (nearly) orthogonal directions in (4.3). Hence by varying λ, we can obtain a wide range of motion. Since it's possible the robot may need to reverse course (particularly in more complicated situations like mazes), we widen the range of allowable λ values to $[-1, 2]$, again sampling ten values over which to optimize.

To demonstrate the efficacy of the algorithm, we chose an $\mathbf{x}_0$ in the first quadrant. The trajectory resulting from (4.2) (with $\beta = 2$) and (4.3) is shown at left in Fig. 4.1. In contrast to the previous graphs, the iterates are now shown only at each t_*. With the addition of the vector $\mathbf{z}$, the robot is able to bypass the obstacle, though not optimally: the r term in F forces the robot to follow a radial path until its proximity to the barrier causes700 it to veer to the left.

At right of Fig. 4.1 we show the evolution of F. Though not clear from the semilog plot, F decays linearly initially until the robot approaches the barrier. Then F remains roughly constant as the robot navigates around the barrier. In this region, precious time is lost as the robot oscillates back and forth to balance the r and barrier portions of F. Then F returns to a linear descent once it turns the corner, only to oscillate as the robot slows down near the target.

Note that compared to Fig. 3.8, the robot takes much longer to come to a stop. For small r, the velocity portion of (4.2) is small enough that it is comparable to the barrier portion, thus confusing the robot about what to do. Therefore, in future work it may

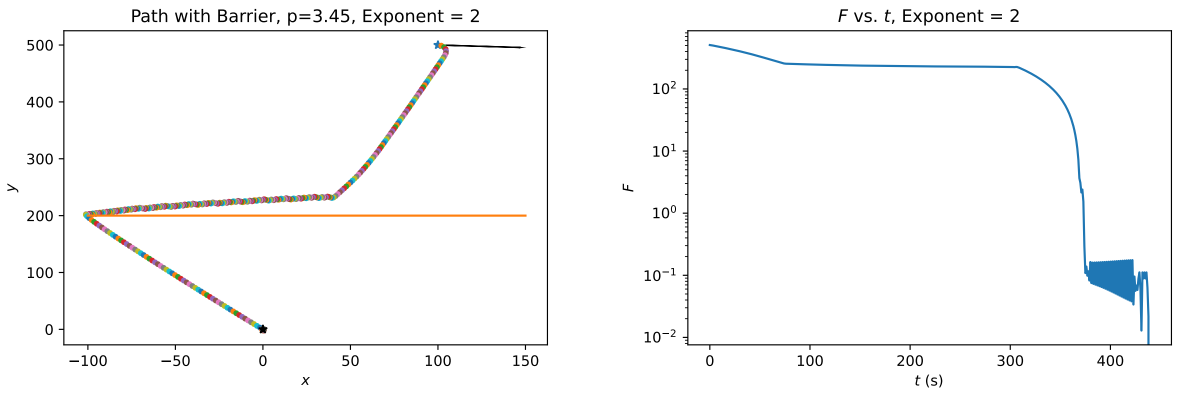


Figure 4.1. Left: Robot path using (4.2) (with $\beta = 2$) and (4.3). Wall is shown as orange line. Right: Value of objective function.

be more useful to keep the barrier term only for $r > r_{\mathrm{s}}$, since the barrier would never impinge upon the target.

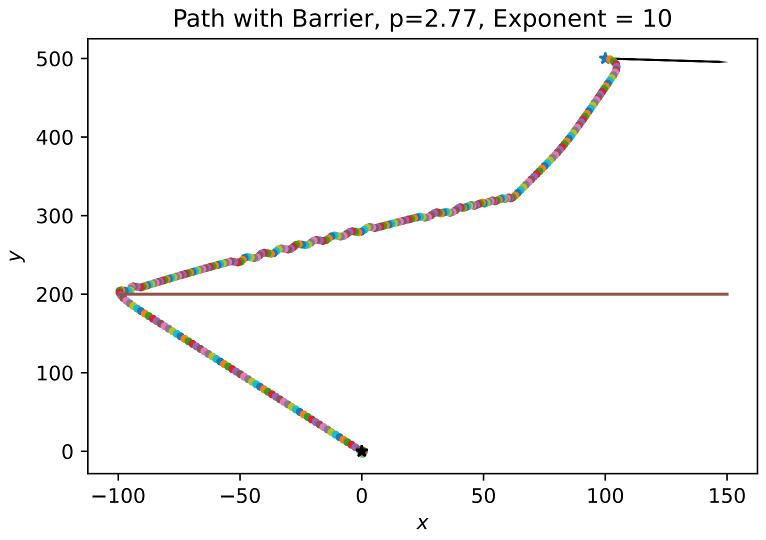


Figure 4.2. Left: Robot path using (4.2) (with $\beta = 10$) and (4.3). Right: Value of objective function.

At first the trajectory in Fig. 4.1 is radial, unnecessarily lengthening the robot's path. Therefore, an increase in the barrier term in F is needed, which we achieve by increasing β in (4.1) from 2 to 10: the results are shown in Fig. 4.2. The trajectory length is reduced, as indicated by the reduced value of p. (Note that the lower bound for the performance factor is no longer 1 due to the obstacle.) One can also discern a “stairstep” pattern in the trajectory once it starts to avoid the barrier. This is most probably a function of the coarse set of λ values the code searches.

4.1 Other Approaches to Obstacles

Increasing the exponent of the barrier term forces the penalty from the barrier to be much larger than the contribution from the radius. Hence the robot prioritizes maneuvering around the obstacle, switching to radius reduction only after the obstacle has been avoided, as shown in Fig. 4.2. To shorten the distance even further, it may be useful to take the limit as the exponent gets large, which is just an exponential:

$$\exp\left(\frac{(x+100)^+}{|y+200|}\right),$$

similar to the forms in Sec. 5.2. Time ran out at the workshop before we could explore this idea more completely.

For other shapes, the basic principle is the same: make the cost function infinite at the barrier. So typical forms would be

$$|\mathbf{r}-\mathbf{r}_\mathrm{o}|^{-\beta}, \quad \beta > 0; \qquad \exp\left(\frac{1}{|\mathbf{r}-\mathbf{r}_\mathrm{o}|}\right),$$

where $\mathbf{r}_\mathrm{o}$ describes the boundary of the obstacle. In addition, the local optimization procedure allows the robot to adapt to changes in its environment, such as new or moving obstacles. As long as the robot has some sensor capability, it can estimate $\mathbf{r}_\mathrm{o}$ at some time t. Then with adequate onboard computing power, it can compute F at that time and map a path around it.

4.2 A More Robust Approach

The fact that $\mathbf{x}$ and $\mathbf{v}$ become aligned severely limits the choice of accelerations the robot can impose using (3.1). The kludge (4.3) is inelegant and unmotivated, but we may implement a more robust approach by returning to first principles. Given the known magnitude a_max of the acceleration, we may choose **only** its direction $\hat{\mathbf{a}}$, which we may write as

$$\hat{\mathbf{a}} = (\cos(\theta+\phi), \sin(\theta+\phi)), \quad \phi \in [0, 2\pi], \tag{4.4}$$

where ϕ measures the change in orientation from the $\mathbf{x}$ vector.

With this change, we can rerun our obstacle code from the previous section, optimizing over ϕ instead of λ. We ensure that two of the sampled values are $\phi = 0$ and $\phi = \pi$, which correspond to $\hat{\mathbf{a}}$ being aligned with $\mathbf{x}$. (We could also use (4.4) for our unobstructed codes, but did not have time to do so at the workshop.)

The trajectory resulting from (4.4) is shown at left in Fig. 4.3. The curve looks similar in shape to Fig. 4.1. Near the wall, the iterates are more widely spaced, indicating that the robot is going faster in this part of the trajectory. It approaches the origin for the first time at around $t = 150$ s, which is when F first reaches a steady state at the right of Fig. 4.3. This is about half the time shown in Fig. 4.1, but all of the oscillations in the F graph after that represent the robot trying to slow itself down, which takes much longer than the corresponding time in Fig. 4.1. Hence the overall p value is abysmal.

This may be remedied using a hybrid approach. Away from the origin, (4.4) seems to do a better job of lowering the travel time. But near the origin, when velocity reduction

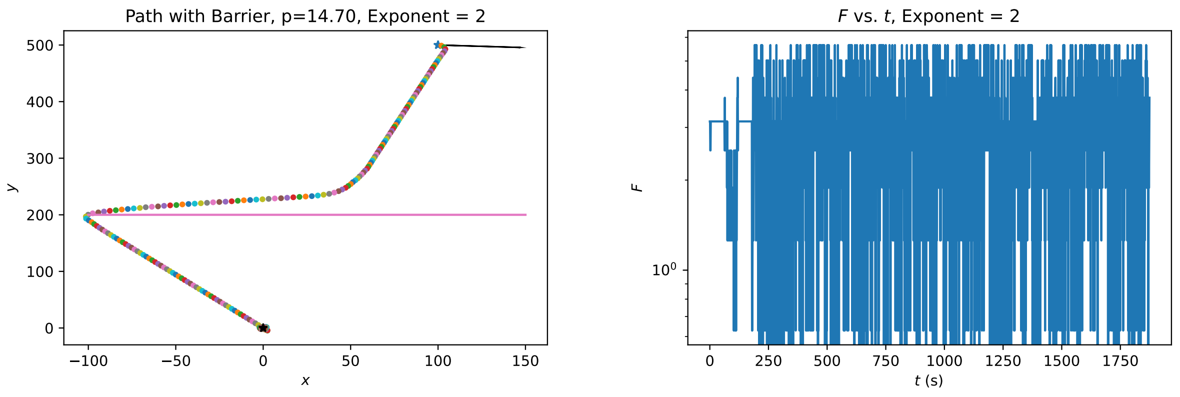


Figure 4.3. Left: Robot path using (4.2) (with $\beta = 2$) and (4.4). Right: Value of objective function.

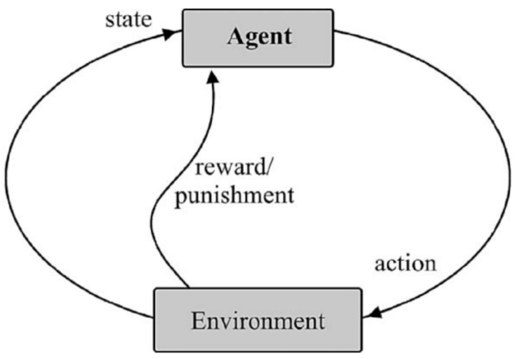


Figure 5.1. A schematic of basis reinforcement learning.

dominates, aligning $\mathbf{a}$ with $\mathbf{v}$ is critical. Hence in that region, we should switch to a control using (3.1). Time ran out before we were able to implement those ideas.

5 Reinforcement Learning

There have been some machine learning studies of this problem (cf. [1]).

Reinforcement Learning (RL) is a class of Machine Learning, which is about learning to make a good sequence of decisions. This means that the algorithm will map the situations into actions. Figure 5.1 shows a diagram to explain the basic structure of a reinforcement learning mechanism. The agent is a goal-seeking object that is trying to observe some parameters from the given environment and takes an action given the current state of parameters. The action will change the current environment and based on the change in the situation, the agent gets a reward or penalty. The goal of the algorithm is to maximize the reward over many iterations. Hence, in order to incentivize desired behavior, an efficient and effective reward function is necessary.

5.1 Current Reward Model

To describe our reward models, we use the subscript i to reflect the value of any quantity at time step i. The current reward model rewards the agent based on its distance to the origin and its relative location within a reward radius, as follows:

$$F_i = \left(\frac{R_{\text{reward}}}{R_{\text{out}}} + R_{\text{reward}} - \mathbf{x}_i \right), \tag{5.1}$$

where R_{reward} is the reward radius and R_{out} is the boundary radius. Here F_i is the reward at step i. Note that we use the same letter as in previous sections for the penalty (cost) function, but they can easily be related with a minus sign. Also, in contrast to the previous sections, where the cost was evaluated at each time step and then discarded, in this section we accumulate the total reward W over time, so

$$W_i = \sum_{k=1}^{i} F_k. \tag{5.2}$$

So as the agent position closes in on the origin, the total accumulated reward increases. As well as this reward, there is an additional penalty which penalizes the agent exiting the boundary radius. The problem with this reward function is that there is no mechanism which incentivizes the agent to reach the origin with as few time steps as possible. To account for this missing time incentive, we have developed three systems to reward desired agent behavior.

5.2 Proposed Models

In the first model, the reward function uses the distance at current and previous step to determine whether the agent has stepped closer to the origin or away from the origin. Therefore, the reward is a function of difference between the previous and current positions:

$$F_i = C(r_{i-1} - r_i).$$

As shown in Fig. 5.2, the green line segments show positive reward because of the step towards the origin. The red line segment shows the penalty for stepping away from the origin. The outer domain and inner domain radii are denoted by R_{out} and R_{in} respectively. The agent gets maximum penalty if the current position is outside R_{out} and gets maximum reward if inside R_{in}.

The second model is similar to the first model but with the reward as a function of area difference. Therefore it is a function of the distance squared:

$$F_i = C(A_{i-1} - A_i) = C'(r_{i-1}^2 - r_i^2).$$

Fig. 5.3 shows the schematic of this reward mechanism. The benefit of using this function is it increases the reward by square of the distance, which makes the agent learn to take larger steps, eventually getting to the origin in less time than the first model.

The third model utilizes an exponential scoring system. We introduce the maximum time limit T and the time step size dt. We introduce four different components to the

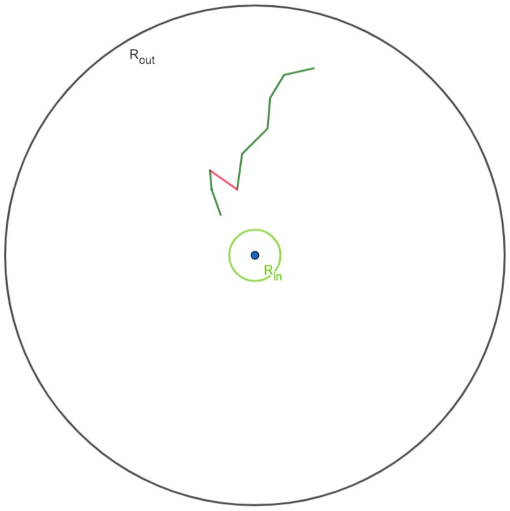


Figure 5.2. Reward mechanism using step direction.

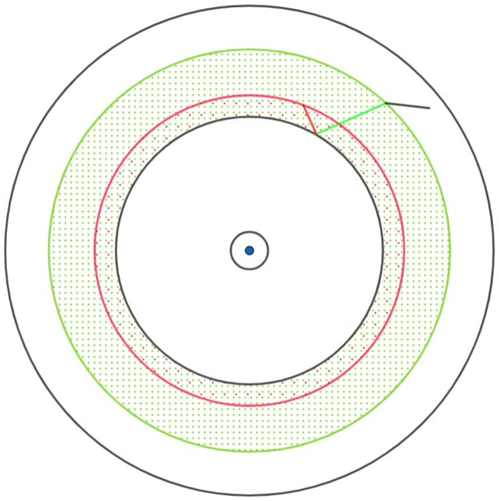


Figure 5.3. Reward mechanism using area difference.

reward, characterized by coefficients b. At each step, we reward the agent if it travels closer to the origin by

$$b_1^{(1-\frac{r_i}{r_{i-1}})(1-\frac{i}{T/dt})}$$

and

$$b_2^{(1-\frac{\theta_i}{\pi/2})(1-\frac{i}{T/dt})}.$$

This rewards the agent for traveling directly to the origin relative to its previous position. As well, once the agent crosses the maximum distance required to slow to a stop at the origin, we give an additional reward of

$$b_3^{(1-\frac{v_i}{v_{\max}})(1-\frac{i}{T/dt})}.$$

Finally, if the agent arrives at the origin, we multiply the total accumulated reward W_i by a bonus time multiplier:

$$b_4^{(1-\frac{i}{T/dt})} W_i.$$

This bonus multiplier, as well as the time terms in the rewards give the agent incentive to complete the iteration as quickly as possible. This reward system should prevent needless wandering by the agent. Unfortunately due to time constraints, we were unable to test this reward system for a significant period of time.

6 Analytical Approaches

In this section, we present several analytical approaches to the posed problem. First, we relax the maximal velocity constraint and the final velocity requirement, and consider the travel time-minimizing problem with only a maximal acceleration. There, we find that the robot should simply travel with maximal acceleration to hit the origin. Then, we consider the problem in which the final position and velocity must be the origin and zero, respectively, under the usual maximal acceleration constraint but without the maximal velocity one.

6.1 Analytical Solution

6.1.1 *Assuming only maximum acceleration*

We are given an initial position $\mathbf{x}_0$ and an initial velocity $\mathbf{v}_0$. Assume we have any curve $\mathbf{x}(t)$ with these initial conditions and a time T such that $\mathbf{x}(T) = \mathbf{0}$. We then have

$$\int_0^T \mathbf{v}(t)\, dt = -\mathbf{x}_0.$$

Let $\mathbf{v}(t) = \mathbf{v}_0 + \mathbf{w}(t)$. Then

$$\int_0^T \mathbf{w}(t)\, dt = -T\mathbf{v}_0 - \mathbf{x}_0 \equiv \mathbf{X}. \tag{6.1}$$

Now let $\mathbf{u}(t) = \mathbf{a}t$ where the vector $\mathbf{a}$ is chosen so that

$$\int_0^T \mathbf{u}(t)\, dt = \mathbf{X}, \tag{6.2}$$

i.e.,

$$\mathbf{a} = \frac{2\mathbf{X}}{T^2}.$$

Observe that if $|\mathbf{a}| > \max_t |\dot{\mathbf{w}}(t)|$ then

$$\frac{\dot{\mathbf{w}}(t) \cdot \mathbf{a}}{|\mathbf{a}|} < |\mathbf{a}|.$$

This implies that $\mathbf{w}(t)\cdot\mathbf{a} < \mathbf{u}(t)\cdot\mathbf{a}$ for all positive times, which then implies

$$\frac{d}{dt}\left(\int_0^t \mathbf{w}(s)\,ds\cdot\mathbf{X}\right) < \frac{d}{dt}\left(\int_0^t \mathbf{u}(s)\,ds\cdot\mathbf{X}\right),$$

which is a contradiction since the two parenthetical expressions are equal at time T.

What the above shows is that any solution $\mathbf{x}(t)$ can be replaced by a solution with constant acceleration without changing the time T it takes to reach the origin and without increasing the maximum acceleration. Thus to find a minimal T, we can restrict to solutions with **constant acceleration**. Therefore we consider

$$\mathbf{a} = \frac{2\mathbf{X}}{T^2}$$

to be a function of T and seek the smallest T for which

$$|\mathbf{a}| = a_{\max}.$$

This is an algebra problem.

6.1.2 *Assuming maximum acceleration and reaching the origin with zero velocity*

First, we provide a reasonable *ansatz* for the expression of the solution. As we saw above, for a given $\mathbf{x}(t)$ over $[0,T]$, (6.1) must hold. Observe now, however, that there is the added constraint $\mathbf{w}(T) = -\mathbf{v}_0$, forcing the final velocity to zero. In this case it is not possible in general to take

$$\mathbf{w}(t) = \mathbf{a}t$$

as we did before. Instead let us write

$$\mathbf{w}(t) = f(t)\mathbf{X} + g(t)\mathbf{v}_0. \tag{6.3}$$

Observe that $f = 2t/T^2$ and $g \equiv 0$ gives us the solution from Sec. 6.1.1. Also, the condition of reaching the origin at time T implies

$$\int_0^T f(t)\,dt = 1, \qquad \int_0^T g(t)\,dt = 0,$$

and having zero velocity at time T implies $f(T) = 0,\ g(T) = -1$. These conditions must hold for any candidate solution $\mathbf{x}(t)$.

For fixed T, there are unique f and g satisfying these conditions belonging to the space of continuous, piecewise linear functions over $[0,T]$, and having minimal slope within this subspace. By arguments similar to those in the first section, we can always replace a solution with these functions without increasing the maximum slopes of each function separately. Below is an example of a path generated using this approach.

If we calculate

$$\dot{\mathbf{w}} = \dot{f}(t)\mathbf{X} + \dot{g}(t)\mathbf{v}_0, \tag{6.4}$$

we find that

$$|\dot{\mathbf{w}}|^2 = \dot{f}(t)^2|\mathbf{X}|^2 + f(t)g(t)\mathbf{X}\cdot\mathbf{v}_0 + \dot{g}(t)^2|\mathbf{v}_0|^2. \tag{6.5}$$

For future work, we will investigate the class of functions to which f and g belong by first assuming $\mathbf{X}\cdot\mathbf{v}_0 = 0$ as a base case (so that the middle term in (6.5) drops out).

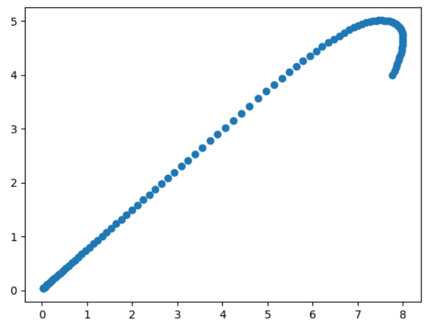


Figure 6.1. Path using algorithm in Sec. 6.1.2.

6.2 Using a Switching Function

Suppose that the velocity and acceleration of the robot are both limited, and suppose that $\mathbf{x}_*(t)$ is an optimal (that is, a quickest) trajectory to the origin (any space dimension is allowed here). We shall show that there is no time interval such that the velocity and acceleration of $\mathbf{x}_*(t)$ are continuous and under **both** the speed and acceleration limits. Thus if an optimal $\mathbf{x}_*(t)$ is piecewise smooth, then on each of the smooth segments $\mathbf{x}_*(t)$ moves at either the limiting top velocity, or top acceleration, or both.

The proof is by contradiction. Assume that there is an interval, which by a shift and rescaling can be taken as an interval $I = [0, 1]$, such that $|\dot{\mathbf{x}}_*(t)| < v_{\max}$, $|\ddot{\mathbf{x}}_*(t)| < a_{\max}$ on I. Since velocity and acceleration are continuous, there is an $\varepsilon > 0$ such that

$$|\dot{\mathbf{x}}_*(t)| < v_{\max} - \varepsilon, \quad |\ddot{\mathbf{x}}_*(t)| < a_{\max} - \varepsilon, \qquad t \in I. \tag{6.6}$$

We claim that the robot can travel on the same trajectory a little faster and still satisfy (6.6). We can show this by introducing a parametrization $t(s)$ for time, where

$$0 < s < S < 1, \quad t(0^+) = 0, \quad t(S^-) = 1. \tag{6.7a}$$

Moreover, if we introduce the constraints

$$\lim_{s\to 0^+} \frac{dt}{ds} = \lim_{s\to S^-} \frac{dt}{ds} = 1, \qquad \lim_{s\to 0^+} \frac{d^2t}{ds^2} = \lim_{s\to S^-} \frac{d^2t}{ds^2} = 0, \tag{6.7b}$$

then $\mathbf{x}_*(t(s))$, as a function of the new parameter s, still satisfies (6.6). If we can satisfy (6.7) with $S < 1$, it takes $\mathbf{x}_*(t(s))$ less time to smoothly traverse the same path as before, and so the original $\mathbf{x}_*(t)$ cannot be optimal. Details are given below; they are not too illuminating, and basically indicate to travel on the same path, but a bit faster, yet not too fast to hit the velocity or acceleration bounds.

To achieve our goal, let $\varphi(s)$ be a smooth switching step function,

$$\varphi(s) = \begin{cases} 0, & s \le 0, \\ \dfrac{1}{2}\left[1 + \tanh\left(\dfrac{2(s - 1/2)}{s(1-s)}\right)\right], & 0 < s < 1, \\ 1, & s \ge 1. \end{cases} \tag{6.8}$$

All the derivatives of the switching function φ are zero at the endpoints $s = 0$, $s = 1$, and the first and second derivatives of it are bounded, so there exists an M such that $|\dot{\varphi}| < M$, $|\ddot{\varphi}| < M$, $0 \le s \le 1$.

Let us use this switching function to switch smoothly from $t(s) = s$ (corresponding to the original velocity) to the new parametrization

$$t(s) = ks, \qquad k = (1+\delta), \qquad 0 < \delta < 1, \tag{6.9}$$

which corresponds to a faster velocity. We impose the switching over some interval $s \in [0, S_1]$, so we have

$$t(s) = s\left[1 - \varphi\left(\frac{s}{S_1}\right)\right] + ks\varphi\left(\frac{s}{S_1}\right), \quad 0 < s \le S_1. \tag{6.10a}$$

We travel at the higher velocity in some interval $s \in [S_1, S_2]$, and hence we require the parametrization (6.9) to hold there. Then we use the switching function again to switch smoothly from $t(s) = ks$ back to $t(s) = s$ over the interval $[S_2, S]$:

$$t(s) = kS_2 + k(s-S_2)\left[1 - \varphi\left(\frac{s-S_2}{S-S_2}\right)\right] + (s-S_2)\varphi\left(\frac{s-S_2}{S-S_2}\right), \quad S_2 < s < S. \tag{6.10b}$$

Then $t(s)$ is smooth, $t(0^+) = 0$, $t(S^-) = S + \delta S_2$, and (6.7b) is satisfied, provided $S + \delta S_2 = 1$, since all derivatives of the switching function are zero at the endpoints.

For specificity, we let

$$S_1 = \frac{S}{4}, \quad S_2 = \frac{3S}{4}, \quad S = \frac{1}{1 + 3\delta/4};$$

the last is to ensure that $t(S^-) = 1$. Since $0 < \delta < 1$, we have that

$$\frac{4}{7} < S = \frac{1}{1+3\delta/4} < 1, \tag{6.11a}$$

which implies that

$$\frac{1}{S_1}, \frac{1}{S - S_2} < 7, \qquad S_1, S_2, S < 1. \tag{6.11b}$$

Moreover, since derivatives of the switching function are bounded, and (6.11) holds, there exists an M_2 such that

$$\left|\frac{dt}{ds}\right| < 1 + M_2\delta, \qquad \left|\frac{d^2t}{ds^2}\right| < M_2\delta. \tag{6.12}$$

Since by the Chain Rule

$$\frac{d\mathbf{x}_*}{ds} = \frac{d\mathbf{x}_*}{dt}\frac{dt}{ds}, \qquad \frac{d^2\mathbf{x}_*}{ds^2} = \frac{d^2\mathbf{x}_*}{dt^2}\left(\frac{dt}{ds}\right)^2 + \frac{d\mathbf{x}_*}{dt}\frac{d^2t}{ds^2}, \tag{6.13}$$

it follows from (6.6) and (6.12) that we can choose δ small enough so that (6.6) is still satisfied, yet it takes less time, namely

$$\frac{1}{1+3\delta/4} < 1,$$

to travel on the same path as before.

6.3 Hamiltonian Approach

In this section, we set up the Hamiltonian for the problem with constraints on both acceleration and velocity. Let $\mathbf{x}(t)$ be the position of the robot at time t. We wish to land at the origin with zero velocity while also minimizing travel time by controlling the acceleration $\mathbf{a}(t)$. This can be posed as a time-optimal control problem:

$$\min_{\mathbf{a}\in\mathcal{U}} \int_0^T dt,$$

where $\mathcal{U}$ is an admissible class of functions. The above must be solved subject to

$$\dot{\mathbf{x}} = \mathbf{v}; \qquad \mathbf{x}(0) = \mathbf{x}_0, \quad \mathbf{x}(T) = 0, \quad \mathbf{v}(0) = \mathbf{v}_0, \quad \mathbf{v}(T) = 0.$$

We must also specify the relationship between $\mathbf{v}$ and $\mathbf{a}$. Though we could incorporate (1.1) in this circumstance, for our purposes we use the same forced constraint as before:

$$\dot{\mathbf{v}} = \mathbf{a}; \qquad |\mathbf{a}(t)| \le a_{\max}, \quad |\mathbf{v}(t)| \le v_{\max}. \tag{6.14}$$

Note that the constraint on $\mathbf{a}$ is device-driven, while the constraint on $\mathbf{v}$ is artificial.

Incorporating the inequality constraint on velocity and acceleration, we define the Hamiltonian

$$\mathcal{H}\left(\mathbf{x},\mathbf{v},\mathbf{a},\vec{\lambda}_1,\vec{\lambda}_2,\mu\right) = 1 + \vec{\lambda}_1^{\mathrm{T}}\mathbf{v} + \vec{\lambda}_2^{\mathrm{T}}\mathbf{a} + \frac{\mu_1}{2}\left(|\mathbf{v}| - v_{\max}\right)^2 + \frac{\mu_2}{2}\left(|\mathbf{a}| - a_{\max}\right)^2, \tag{6.15}$$

where the $\vec{\lambda}_i$ are Lagrange multipliers of length equal to the dimension of the problem and the $\mu_i \ge 0$ are scalar Lagrange multiplier that enforce the inequality constraints in (6.14). Note that if $\mu_1 > 0$, we must have $|\mathbf{v}| = v_{\max}$ which means we are on the boundary of the velocity constraint (saturation). If $\mu_1 = 0$, the velocity is strictly less than the constraint, which is mathematically equivalent to the problem without a velocity constraint at all. We may use similar reasoning for the constraint on $|\mathbf{a}|$.

From here on, we solve the problem in one dimension, *i.e.*, the Lagrange multipliers are scalars. Before we start, we note that having $\mu_2 = \lambda_2$ will yield the same optimal solution. To see this, we minimize $\mathcal{H}$ pointwise in time first. When $\mu_1 = 0$ (when the velocity is strictly less than the constraint), we find that the pointwise optimizer $\tilde{a}$ is given by

$$\tilde{a}(x,v,\lambda_1,\lambda_2) = \begin{cases} -a_{\max}\mathrm{sgn}(\lambda_2), & \lambda_2 \ne 0, \\ [-a_{\max}, a_{\max}], & \lambda_2 = 0. \end{cases}$$

Clearly, when $\mu_1 > 0$, in order to minimize the Hamiltonian, we must also maintain the velocity at $v_{\max}$. This implies that we should solve the problem with $\mu_1 = 0$ and saturate the velocity constraint whenever it is reached. We also point out that when $\lambda_2 \ne 0$, we must have $a = \pm a_{\max}$, which automatically kills the acceleration constraint in (6.15) (and helps minimize $\mathcal{H}$), thus justifying the choice that $\mu_2 = \lambda_2$.

Denote $a_*(t)$ as the optimal control. At $a = a_*$, the costate equations are

$$\begin{aligned} \dot{\lambda}_1(t) &= -\frac{\partial\mathcal{H}}{\partial x} = 0, \\ \dot{\lambda}_2(t) &= -\frac{\partial\mathcal{H}}{\partial v} = -\lambda_1(t). \end{aligned}$$

This implies

$$\lambda_1(t) = \lambda_1^0, \quad \lambda_2(t) = -\lambda_1^0 t + C.$$

We discuss several possibilities of the λ_i's. First, we know from optimal control theory that if the Hamiltonian does not depend on time explicitly, then $\mathcal{H}\left(x, v, a^*, \lambda_1, \lambda_2, \mu\right) = 0$ (evaluated at optimum a^*). This implies that λ_1 and λ_2 are both never zero, or otherwise $\mathcal{H}\left(x, v, a^*, \lambda_1, \lambda_2, \mu\right) = 1$ which is a contradiction. Therefore, both costates are never zero.

Altogether, we must have $a_*(t) = \pm a_{\max}$, evident from the fact that λ_2 is a linear function in time, which implies that it changes sign only once. Using the acceleration, we then have

$$x(t) = \pm \frac{a_{\max}}{2} t^2 + v_0 t + x_0, \tag{6.16a}$$

$$v(t) = \pm a_{\max} t + v_0. \tag{6.16b}$$

We emphasize that there are two particular curves represented by (6.16) that pass through the origin of the phase plane, *i.e.*, $(x, v) = (0, 0)$. They are important because they are optimal paths that also satisfy the final condition, namely, zero velocity at the origin (of position). We call these two curves *switching curves*. Once the position and velocity of the robot satisfy the expression of the switching curves, the robot should immediately travel according to them. To obtain their expressions, we first relate $x(t)$ and $v(t)$,

$$x(t) = \pm \frac{1}{2a_{\max}} v^2(t) + x_0 \mp \frac{v_0^2}{2a_{\max}}. \tag{6.17}$$

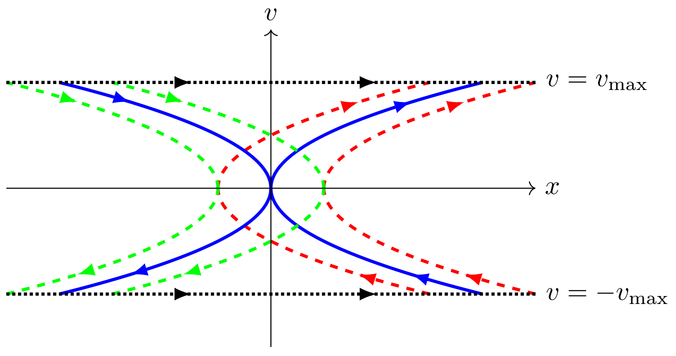
The switching curves then satisfy

$$x(t) = \pm \frac{1}{2a_{\max}} v^2(t) \tag{6.18}$$

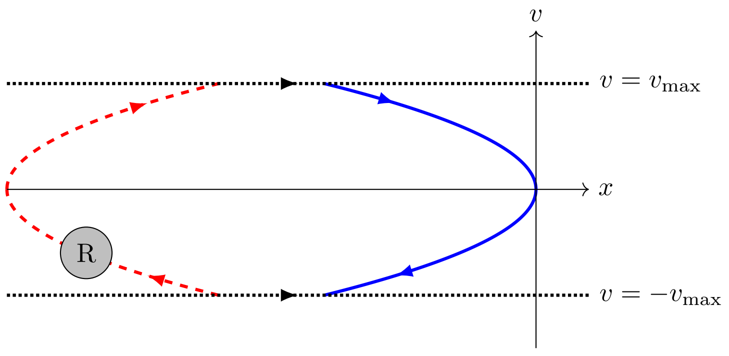
since they pass through $(x, v) = (0, 0)$. This set of trajectories can be represented in an x-v phase plane (see Fig. 6.2).

To incorporate the velocity constraint $|v(t)| \leq v_{\max}$, we simply place two horizontal lines at $v = \pm v_{\max}$, respectively. We describe the optimal path of the robot as follows. Starting with an initial position x_0 and velocity v_0, the robot travels on an optimal trajectory within the velocity and acceleration constraints described by (6.17). It encounters two mutually exclusive scenarios: 1) hits a switching curve (6.18) before obtaining maximal velocity or 2) accelerates to maximal velocity without hitting any of the switching curves. In the former, the problem would be the same as if there is no maximal velocity constraint ($\mu = 0$); the robot goes on the switching curve and heads to origin, or in other words, the control is *bang-bang*. In the latter, the robot will travel at maximal velocity until it reaches a switching curve; this control is known as *bang-off-bang* since the robot initially travels with a nonzero acceleration, releases the gas pedal and then steps on it again. We note that it is possible that after reaching the maximal velocity, the robot never visits a switching curve. This implies that there are certain initial conditions under which there are no optimal solutions.

This 1D problem with both velocity and acceleration constraints is fully solved by Fehér *et al.* [2], who considered a continuous-time double-integrator (*i.e.*, second order)



(a)



(b)

Figure 6.2. (a) The phase plane: blue solid lines are the switching curves (6.18); red and green dashed lines are optimal trajectories when the maximal velocity and acceleration constraints are not violated; black dotted lines are the maximal velocity allowed.
(b) An example for $x(t) < 0$: red dashed line is one particular optimal trajectory within the velocity and acceleration constraints following (6.17) with some initial position and velocity. Robots (such as the lightgray shaded ball) traveling on the this curve encounter a bang-off-bang control, that is, 1) (*bang*) accelerates (at $a_{\max}$ in this case) until it hits the maximal speed allowed $v_{\max}$ (black dotted), 2) (*off*) travels at this velocity until 3) (*bang*) it reaches one of the switching curves (solid blue) and decelerates into the origin (at $-a_{\max}$ in this case).

system. The control function (acceleration) takes values $\{-a_{\max}, 0, a_{\max}\}$, under conditions on position and velocity.

7 Conclusions and Further Research

7.1 Summary of Results

At first glance, the problem of guiding a robot with initial position $\mathbf{x}_0$ and velocity $\mathbf{v}_0$ to a stop at the origin seems quite simple. However, upon further investigation, the problem turns out to be quite rich, and amenable to a variety of different solution techniques.

We considered computational optimization, reinforcement learning, and analytical approaches.

The variational approaches show that under certain specified conditions it is possible to find an absolute minimizer. However, extending this approach to more complicated problems, such as the presence of obstacles, may be difficult. For the computational optimization and reinforcement learning approaches, we can consider adjusting the reward functions to account for obstructions. We speculate that the computational optimization model will be easier and more cost efficient to implement than the reinforcement learning model. However, an underlying assumption within the computational optimization model is that the optimal action in the local environment is the optimal action for global environment. The reinforcement learning model does not make this assumption and instead focuses on the optimal sequence of actions for the global environment.

7.2 Further Research

With the limited time afforded to us during the workshop, we were unable to investigate many ideas that may improve our results. Some of these have been outlined in the individual sections above; we conclude by listing some other more general ideas.

All of the computational trajectories exhibit evidence of overshoot around the origin. This may be due to the fact that we never change the magnitude of the acceleration. To remedy this, we could try the following form:

$$|\mathbf{a}| = \min\left\{a_{\max}, \frac{v^2}{2r}\right\}, \tag{7.1}$$

where the second term in (7.1) is motivated by a rearrangement of (2.4). There may still be some problems due to the singular nature of the fraction near the origin; we did not have time at the workshop to investigate (7.1) in detail.

When there were no obstacles, each trajectory quickly aligned $\mathbf{v}$ and $\mathbf{x}$ and approached the origin on a radial path. It may be useful to create that as an objective, and cause the acceleration to align the two vectors first before driving the robot to the origin.

Each of the methods were able to produce results that solved the problem. The local optimization and machine learning methods rely upon the selection of an appropriate function to be optimized. For future research, it would be beneficial to test the proposed reward functions more rigorously with an eye toward improving them.

For the variational principles approach, an investigation of a hodograph transformation [5, p. 182] may be useful. This is a transformation used in PDEs which switches the dependent and independent variables. (Hence t would be come a dependent variable, and might be easier to optimize.) This idea has yet to be pursued.

Nomenclature

Units are listed in terms of length (L) and time (T). Equation numbers where a variable is first defined is listed, if appropriate.

A: area, units L^2.
$\mathbf{a}$: acceleration, units L/T^2.

a: magnitude of acceleration, units L/T^2 (2.1).
b: parameter in exponential cost function.
C: constant, variously defined.
F: cost or reward function.
f: function in definition of $\mathbf{w}$ (6.3).
g: function in definition of $\mathbf{w}$ (6.3).
$\mathcal{H}$: Hamiltonian (6.15).
i: indexing variable for time step (5.1).
j: indexing variable for time step (5.2).
k: scaling factor for parametrization (6.9).
M: bound in switching function section.
N: number of iterations.
p: performance factor (3.3).
R: radial boundary in reward function, units L (5.1).
$\hat{\mathbf{r}}$: unit vector in the radial direction (2.5a).
r: distance of robot from origin, units L (2.1).
S: specific value of s (6.7).
s: paremetrization variable (6.7).
T: maximum time, units T.
t: time, units T (2.1).
$\mathcal{U}$: space of feasible accelerations.
$\mathbf{u}$: velocity with constant acceleration, units L/T (6.2).
$\mathbf{v}$: velocity of robot, units L/T.
v: speed of robot, units L/T.
W: accumulated reward (5.2).
$\mathbf{w}$: displacement of velocity from initial state, units L/T (6.1).
$\mathbf{X}$: straight-line displacement, units L (6.1).
$\mathbf{x}$: displacement of robot, units L.
x: Cartesian coordinate (4.1) or one-dimensional position of robot, units L.
y: Cartesian coordinate, units L (4.1).
$\mathbf{z}$: kludged orthogonal direction (4.3).
α: alignment factor (3.2).
β: exponent in cost function (4.1).
δ: bound in switching section (6.9).
ε: used to establish bounds (6.6).
θ: angular coordinate (2.5a).
λ: parameter to adjust acceleration (3.1) or Lagrange multiplier (6.15).
μ: Lagrange multiplier (6.15).
ν: coefficient in friction law, units T^{-1} (1.1).
τ: time measured from start of deceleration, units T (2.2a).
ϕ: angle describing orientation of $\hat{\mathbf{a}}$ (4.4).
$\varphi(s)$: switching step function (6.8).

Other Notation

d: as a subscript, used to refer to the deceleration period (2.2a).
max: as a subscript, used to represent a maximum value.
o: as a subscript, used to represent the obstacle.
s: as a subscript, used to represent a stopping criterion.
0: as a sub- or superscript, refers to an initial condition.
$*$: as a subscript, refers to an acceleration interval or optimal value.
$\dot{}$: used to indicate a derivative with respect to t (2.1).
$\hat{}$: used to indicate a unit vector (2.5a).
$+$: as a superscript, refers to a ramp function (4.1).
$\tilde{}$: used to indicate a pointwise optimizer.

Acknowledgement

This work was supported in part by National Science Foundation grant DMS-2016095, as well as the SIAM James Crowley Fund for Student Support. The authors thank the reviewer for helpful suggestions which greatly improved the manuscript.

References

[1] Bozek, Pavol, Karavaev, Yury L, Ardentov, Andrey A, & Yefremov, Kirill S. (2020). Neural network control of a wheeled mobile robot based on optimal trajectories. *International journal of advanced robotic systems*, **17**(2), 1–10.

[2] Fehér, Marek, Straka, Ondřej, & Šmídl, Václav. (2017). Constrained time-optimal control of double-integrator system and its application in MPC. *Journal of physics: Conference series*, **783**, 012024.

[3] Thornton, S.T., & Marion, J.B. (2021). *Classical dynamics of particles and systems.* Cengage Learning.

[4] Urone, P.P., Hinrichs, R., Dirks, K., & University, Rice. (2012). *College physics.* OpenStax College, Rice University.

[5] Whitham, G.B. (2011). *Linear and nonlinear waves.* Pure and Applied Mathematics: A Wiley Series of Texts, Monographs and Tracts. Wiley.

Appendix A Parameter Values

For the device parameters, we were given

$$v_{\max} = 4\,\frac{\text{m}}{\text{s}}, \qquad a_{\max} = 2\,\frac{\text{m}}{\text{s}^2}. \tag{A 1}$$

For the values of $\mathbf{x}_0$, we were given a bound of

$$x_{\max} = 10^3 \text{ m}. \tag{A 2}$$

The prescribed termination tolerances are given by

$$r_{\text{s}} = 1 \text{ m}, \qquad v_{\text{s}} = 10^{-2}\,\frac{\text{m}}{\text{s}}, \tag{A 3}$$

and we said the algorithm failed if it took more than

$$N_{\max} = 10^5 \tag{A 4}$$

iterations.

For our time intervals, we used

$$dt = 0.01 \text{ s}, \qquad t_* = 1 \text{ s}. \tag{A 5}$$