Time-optimal Trajectories for Holonomic Drive Robots

Eric Wieser

Holonomic or omnidirectional robots are the most common platform used in the Robocup Soccer Small Size League, and benefit from the system's high maneuverability. This maneuverability comes at the expense of complexity, and as such many of the control implementations use abstractions that limit performance, leading to a slower trajectory. In this paper, the dynamics of the system are analyzed and modelled in the Drake control toolbox, and trajectory optimization used to investigate the improvements gained with an optimal controller.

Introduction

Robocup Soccer Small Size League (SSL) is a competition that simulates a football game, where two teams of six robots play with an orange golf ball, using information from overhead cameras. Robot dimensions are constrained to fit within a small cylinder, and further restrictions lead to a design that is capable of receiving the ball at only one location on the perimeter.

Since the game is dynamic and fast-paced, robots need to be able to change direction quickly. This is something that a two-wheeled differential drive is unable to do, as they are unable to exert a force parallel to their wheel axle. Holonomic drives allow the resultant drive force to point in any direction, and change instantaneously in direction (ignoring constraints from the actuators themselves). Additionally, rotational torque can be decoupled from linear acceleration, allowing rotation simultaneously translation – useful for shielding or receiving a ball.

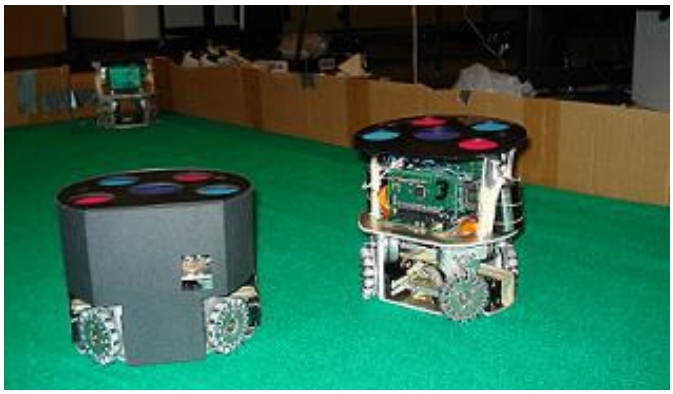


Figure 1 - An old photo of the RFC cambridge robocup robots [2]

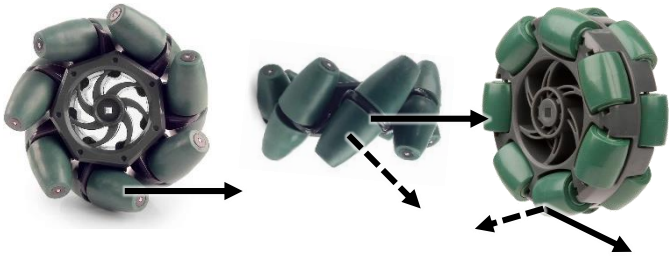
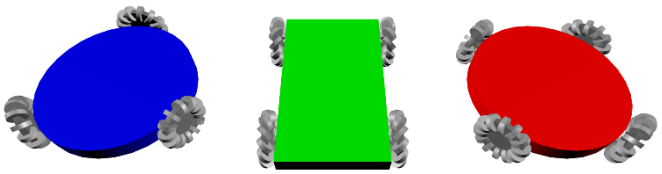


Figure 1 - From left to right, two views of a Mecanum wheel, and an omniwheel, both made by Vex Robotics. The solid arrow is $\vec{e}^{drive}$, and the dashed arrow $\vec{e}^{slip}$, parameters described in Table 1. We refer to the green components as "sub-wheels".

Holonomic Drives

The defining feature of a holonomic drive is its wheels, which consist of one large motor-driven hub wheel, with a set of smaller unpowered sub-wheels whose axes of rotation lie in the tangent plane of the hub. These wheels come in two common forms – "omniwheels", where the sub-wheel axis is orthogonal to the hub axis; and "Mecanum wheels", where the angle between them is around 45°. The former is the type used in Robocup, while the latter has applications in heavier robots, due to details of the mechanical design.

With three or more of these wheels, the kinematics of the robot is fully-defined by the kinematics of the wheels. For Robocup SSL, typically four wheels are used, due to the increased total actuator power, and geometric constraints imposed by the kicking mechanism. Some example drive configurations are shown in Figure 2.



(a) 3 omniwheels *(b) Mecanum wheels* *(c) 4 omniwheels*

Figure 2 - Typical holonomic drive configurations, rendered with Three.JS [3].

Symbol	Quantity	Comments
m	Overall mass	Lumped with wheel mass for simplicity
I	Overall inertial	In the vertical axis. Wheel inertia is not modelled, as it is comparatively small.
N	Number of wheels	State is under-constrained if $N < 3$
	For each wheel:	Vectors measured in the body frame
r_i	Radius	From large wheel axle to ground
$\vec{d}_i$	Location	Relative to the center of mass
$\vec{e}_i^{\text{drive}}$	Drive direction	The unit direction vector in which a linear velocity causes only motor shaft rotation – should be perpendicular to the motor shaft
$\vec{e}_i^{\text{slip}}$	Slip direction	The unit direction vector in which a linear velocity causes only sub-wheel rotation – should be perpendicular to the shaft of the sub-wheel in contact with the ground plane. For "omniwheels", $\vec{e}_i^{\text{drive}} \perp \vec{e}_i^{\text{slip}}$
$\tau_i^{\max}$	Motor stall torque	
$\omega_i^{\max}$	Motor free-running angular velocity	

Table 1 – Parameters of the holonomic drive model

Modelling dynamics

We model the system as a time-invariant second-order state-space system, parameterized as described in Table 1. The model is of the form

$$\vec{q} = [x \quad y \quad \theta]^T, \qquad \ddot{\vec{q}} = f(\dot{\vec{q}}, \vec{q}, \vec{u}),$$

where x, y is the position of the robot in the global frame, and θ is the angle from the world x axis to the robot x axis, measured in the right-hand sense. $\vec{u}$ is a vector of non-dimensional motor voltages, where $|u_i| \leq 1$.

We make a number of simplifying modelling assumptions:

- The inertia of the wheels is neglected
- The sub-wheel axles are frictionless
- All wheels remain in non-slip contact with the ground

We model the motors using a simplified model parameterized by the stall-torque and free-running angular velocity:

$$\tau_i = \tau_i^{\max}\left(u_i - \frac{\omega_i}{\omega_i^{\max}}\right)$$

This requires the angular velocity of the wheel to be found. Using rigid body dynamics, the velocity of the wheel contact point in the body frame can be found as:

$$\vec{v}_i = \vec{v}|_{\text{body}} + \dot{\theta} \times \vec{d}_i$$
$$\vec{v}|_{\text{body}} = R(\theta)^{-1}\vec{v} = \begin{bmatrix}\cos\theta & -\sin\theta \\ \sin\theta & \cos\theta\end{bmatrix}\begin{bmatrix}\dot{x} \\ \dot{y}\end{bmatrix}$$

Here we introduce a 2D cross-product shorthand notation,

$$a \times \vec{v} \stackrel{\text{def}}{=} a\vec{e}_z \times (v_x\vec{e}_x + v_y\vec{e}_y)$$
$$\vec{v} \times \vec{w} \stackrel{\text{def}}{=} (v_x\vec{e}_x + v_y\vec{e}_y) \times (w_x\vec{e}_x + w_y\vec{e}_y) \cdot \vec{e}_z$$

Splitting the velocity into wheel and sub-wheel components,

$$\vec{v}_i = r_i\omega_i\vec{e}_i^{\text{drive}} + v_s\vec{e}_i^{\text{slip}} = \begin{bmatrix}\uparrow & \uparrow \\ \vec{e}_i^{\text{drive}} & \vec{e}_i^{\text{slip}} \\ \downarrow & \downarrow\end{bmatrix}\begin{bmatrix}r_i\omega_i \\ v_s\end{bmatrix}$$
$$\Rightarrow \omega_i = \begin{bmatrix}\frac{1}{r} & 0\end{bmatrix}\begin{bmatrix}\uparrow & \uparrow \\ \vec{e}_i^{\text{drive}} & \vec{e}_i^{\text{slip}} \\ \downarrow & \downarrow\end{bmatrix}^{-1}\vec{v}_i$$

The force exerted on the wheel by the ground must act perpendicularly to $\vec{e}_i^{\text{slip}}$, since the sub-wheels cannot bear tangential loads:

$$\vec{f}_i|_{\text{body}} = f_i|_{\text{body}}(\vec{e}_i^{\text{slip}})^{\perp}$$

Expressing the total torque exerted by the motor in 3D, where α and β are the reaction torques fixing the motor shaft:

$$\vec{\tau}_i = \tau_i(\vec{e}_i^{\text{drive}})^{\perp} + \alpha\vec{e}_z + \beta\vec{e}_i^{\text{drive}}$$

Resolving moments on the wheel in 3D, we have:

$$\vec{\tau}_i = \vec{r} \times \vec{f}_i|_{\text{body}} = -r_i\vec{e}_z \times f_i|_{\text{body}}(\vec{e}_i^{\text{slip}})^{\perp}$$
$$= r_i f_i|_{\text{body}}\vec{e}_i^{\text{slip}}$$
$$\Rightarrow \vec{\tau}_i \cdot (\vec{e}_i^{\text{drive}})^{\perp} = \tau_i = r_i f_i|_{\text{body}}\vec{e}_i^{\text{slip}} \cdot (\vec{e}_i^{\text{drive}})^{\perp}$$
$$\Rightarrow \vec{f}_i|_{\text{body}} = \frac{\tau_i}{r_i}\frac{(\vec{e}_i^{\text{slip}})^{\perp}}{\vec{e}_i^{\text{slip}} \cdot (\vec{e}_i^{\text{drive}})^{\perp}}$$

Note this has a singularity when $\vec{e}_i^{\text{slip}} \parallel \vec{e}_i^{\text{drive}}$, but in this case the wheels are degenerate anyway – the system would act like a ball bearing, exerting no net force

Finally, we use momentum balances and a frame conversion to conclude that:

$$\begin{bmatrix}\ddot{x} \\ \ddot{y}\end{bmatrix} = \frac{1}{m}R(\theta)\sum \vec{f}_i|_{\text{body}}, \qquad \ddot{\theta} = \frac{1}{I}\sum \vec{d}_i \times \vec{f}_i|_{\text{body}}$$

$$\begin{bmatrix}\ddot{x}\\ \ddot{y}\end{bmatrix} = \frac{1}{m}R(\theta)\sum\frac{\tau_i^{\max}}{r_i}\left(u_i - \begin{bmatrix}\frac{1}{r\omega_i^{\max}} & 0\end{bmatrix}\begin{bmatrix}\uparrow & \uparrow\\ \vec{e}_i^{\text{drive}} & \vec{e}_i^{\text{slip}}\\ \downarrow & \downarrow\end{bmatrix}^{-1}\left(R(\theta)^{-1}\begin{bmatrix}\dot{x}\\ \dot{y}\end{bmatrix} + \dot{\theta}\times\vec{d}_i\right)\right)\frac{\left(\vec{e}_i^{\text{slip}}\right)^{\perp}}{\vec{e}_i^{\text{slip}}\cdot\left(\vec{e}_i^{\text{drive}}\right)^{\perp}}$$

$$\ddot{\theta} = \frac{1}{I}\sum\frac{\tau_i^{\max}}{r_i}\vec{d}_i\times\left(\left(u_i - \begin{bmatrix}\frac{1}{r\omega_i^{\max}} & 0\end{bmatrix}\begin{bmatrix}\uparrow & \uparrow\\ \vec{e}_i^{\text{drive}} & \vec{e}_i^{\text{slip}}\\ \downarrow & \downarrow\end{bmatrix}^{-1}\left(R(\theta)^{-1}\begin{bmatrix}\dot{x}\\ \dot{y}\end{bmatrix} + \dot{\theta}\times\vec{d}_i\right)\right)\frac{\left(\vec{e}_i^{\text{slip}}\right)^{\perp}}{\vec{e}_i^{\text{slip}}\cdot\left(\vec{e}_i^{\text{drive}}\right)^{\perp}}\right)$$

Equation 1 - full system dynamics

Combining all these equations gives Equation 1, from it can be found that the dynamics are linear in $\dot{x}, \dot{y}, \dot{\theta}, u_i$, but non-linear in θ.

Parameter values

The parameters below describe the robocup robots as used by RFC Cambridge.

Symbol	Value	
m	1	kg
I	0.0031	kg m^2
N	4	
θ_i	$\left\{\frac{\pi}{4}, \frac{3\pi}{4}, -\frac{3\pi}{4}, -\frac{\pi}{4}\right\}$	rad
r_i	0.025	m
$\vec{d}_i$	$0.078\begin{bmatrix}\cos\theta_i\\ \sin\theta_i\end{bmatrix}$	m
$\vec{e}_i^{\text{drive}}$	$\begin{bmatrix}-\sin\theta_i\\ \cos\theta_i\end{bmatrix}$	
$\vec{e}_i^{\text{slip}}$	$\begin{bmatrix}\cos\theta_i\\ \sin\theta_i\end{bmatrix}$	
$\tau_i^{\max}$	0.75	Nm
$\omega_i^{\max}$	150	rad s^{-1}

Table 2 - parameter estimates for the RFC robot

These were determined from a mixture of estimation and interpretation of constants in existing source code. Unfortunately, access to a fully-assembled robot was not possible during this project.

Basic control implementation

A typical control implementation for this platform implements velocity control on this system by resolving the target angular velocity into rotor angular velocities, and using an isolated PID controller to regulate the velocity of each motor individually.

From this, an implementation of position control can be formed using a 2D equivalent of “bang-bang” or “on-off” control, which simply chooses an achievable velocity pointing directly towards the goal. Typically, orientation will be regulated to remain constant

This controller has a problem – the maximum robot velocity is non-uniform in direction. Consider the robot in Figure 1c moving upwards, with uniform motor parameters. To do this, the control inputs become $\vec{u} = [u_{tl} \quad u_{bl} \quad u_{br} \quad u_{tr}]^T = [1 \quad 1 \quad -1 \quad -1]^T$. This gives us a terminal velocity of $\sqrt{2}r\omega^{\max}$. Contrast this with a movement up and to the right at 45°, which requires $\vec{u} = [1 \quad 0 \quad -1 \quad 0]^T$. This gives us terminal velocity of $r\omega^{\max}$. In general, the terminal velocity vector is constrained to lie in a convex polygon anchored rotationally to the frame of the robot.

A similar argument can be made about acceleration. For the 4-wheeled symmetric robot, the bounding polygon for accelerations is geometrically similar to the one for velocities, but this is not generally true. Later on, we will briefly explore exploiting situations when these differ.

Therefore, we should expect an optimal controller to reorient the robot such that the direction of maximum velocity is aligned with the direction of motion.

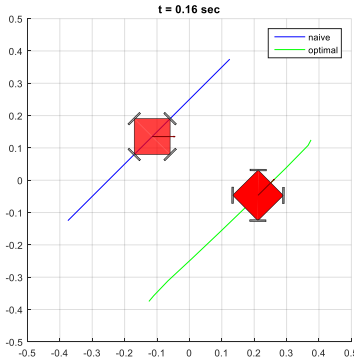


Figure 3 – Comparison of the basic control strategy and an optimal one for the 4-wheel omniwheel drive, showing robot reorientation. Both robots started moving simultaneously, and the optimal controller can be seen to have covered more ground.

Trajectory optimization

To find optimal controllers for a given trajectory, we use direct collocated trajectory optimization. This finds a trajectory composed of concatenated splines, such that the gradient at both the knot-points and the mid-spline matches that described by the system dynamics. To implement this, we use the `DircolTrajectoryOptimization` class within the Drake [1] toolbox.

Formally, we solve the problem:

$$\begin{aligned} \text{find } \vec{x}(t[\cdot])\ s.t. \quad & \vec{x} \text{ satisfies dynamics constraints} \\ & |\vec{u}(t[\cdot])|_\infty < 1 \\ & \vec{x}(t[1]) = \vec{x}_{\text{start}} \\ & \vec{x}(t[N]) = \vec{x}_{end} \\ \text{maximizing} \quad & t \end{aligned}$$

Since we are evaluating the effectiveness of position control, $\vec{x}_{\text{start}}$ and $\vec{x}_{\text{end}}$ were chosen such that the corresponding $\dot{\vec{q}} = 0$.

To evaluate the improvement offered by the reorientation strategy, we should also run an optimization which is constrained to not use this strategy. We desire that

$$\vec{q}_i = \vec{q}_{\text{start}} + \lambda_i(\vec{q}_{\text{end}} - \vec{q}_{\text{start}})\ \forall i,$$

where λ_i is arbitrary constant. In other words, the non-derivative states are constrained to be a linear interpolation between the start and end point. While we could express this constraint by making λ_i a decision variable, this would be a poor choice, as we end up with further non-convexities. Instead, we can express this constraint as a linear one, in matrix form, with some manipulation:

$$\begin{aligned} A\vec{q}_i &= A\vec{q}_{\text{start}} + A\lambda_i(\vec{q}_{\text{end}} - \vec{q}_{\text{start}}) \\ \text{Choose A} : A(\vec{q}_{\text{end}} - \vec{q}_{\text{start}}) &= \vec{0} \\ \Rightarrow A\vec{q}_i &= A\vec{q}_{\text{start}} \end{aligned}$$

We can choose an appropriate A in Matlab using `null((qend - qstart)')'`, where `null(v')'` finds a matrix whose left null space is v.

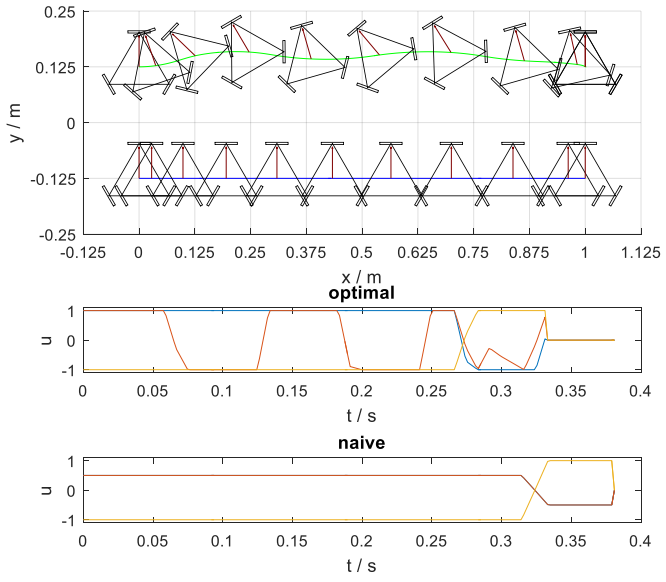


Figure 4 - Trajectory optimization applied to a 3-wheel robot. The upper plot shows the optimal controller in green. The lower two plots show the input signals for the two controllers

Figure 4 shows the result of running this trajectory for a destination 1m away. The optimal controller reaches its target in 0.34s, whereas the naïve controller takes 0.38s – roughly a 10% improvement. Of note is that the optimal controller is running all motors at full power until it approaches its target, whereas the naïve controller leaves motors at half power.

Reachable space

In the previous section, we were optimizing for a fixed trajectory in minimal time. Another interesting question is to optimize for a given time, and find the region of all reachable positions by the robot. This has applications such as determining whether a robot is able to intercept a ball. We restrict the search space to the three dimensional region parameterized by robot position and time.

To conduct this search, we discretize the space into time step and direction vectors. For each horizon time t_{end} and direction vector $\vec{d}$, we run a trajectory optimization to find the furthest reachable location in a given direction

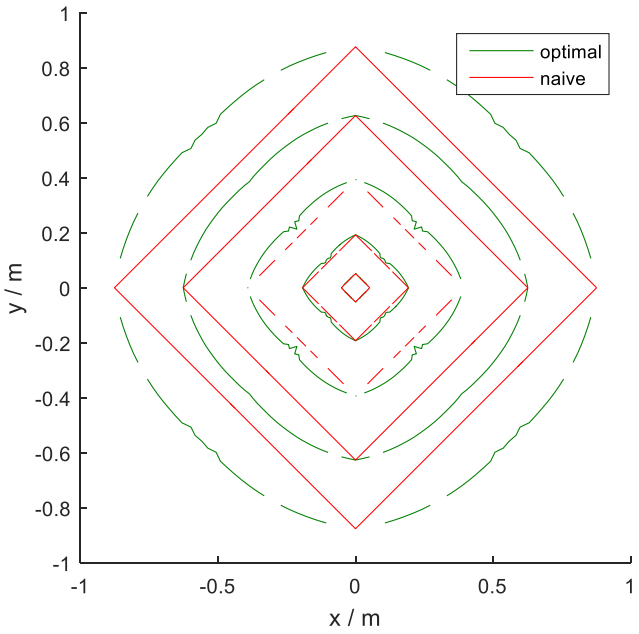
$$\begin{aligned} \text{find } \vec{x}(t[\cdot])\ s.t. \quad & |\vec{u}(t[\cdot])|_\infty < 1 \\ & \vec{x}(t[1]) = \vec{x}_{\text{start}} \\ & t[N] = \vec{t}_{end} \\ & \vec{x}(t[N]) = \vec{x}_{\text{start}} + \lambda\vec{d} \\ & \vec{x} \text{ satisfies dynamics constraints} \\ \text{maximizing} \quad & (\vec{x}(t[N]) - \vec{x}_{\text{start}}) \cdot \vec{d} \end{aligned}$$

Using the same technique as before to re-express the λ constraint. Again, we compare the result of this optimization with the result of running the same optimization with the additional constraints imposed by the simple controller. The results are shown in Figure 5.

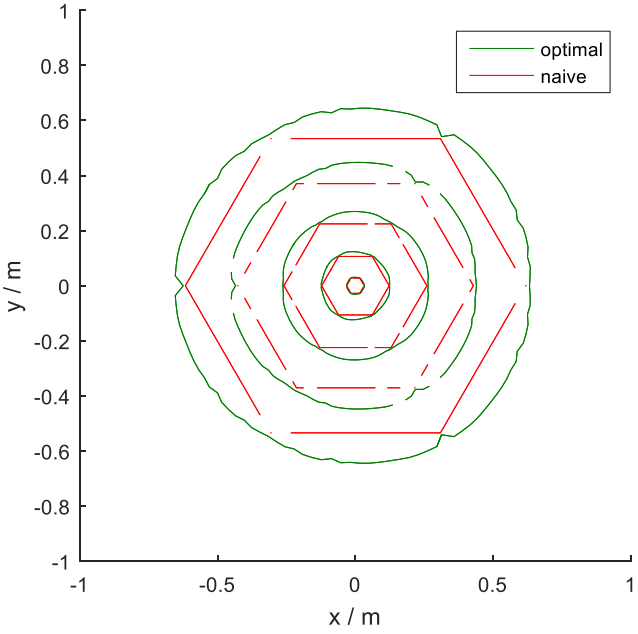
The naïve controller shows the polygonal nature previously predicted. For small time periods, the optimal controller has a very similar horizon, as there is not enough time to reorient. For large time periods, the optimal horizon approaches a circle.

These horizons also show the benefit of choosing a 4-wheel base over a three-wheel base, as for the same robot and motor parameters, the horizon is greater. Of course, this ignores considerations such as incremental motor and battery mass.

There's an interesting effect here for the three-wheeler though: while the optimal controller horizon for the four-wheeler is coincident with the naïve one at the vertices of the polygon, the optimal horizon for the 3-wheeler exceeds it – i.e. there is a yet more efficient form of motion than that of reorienting to the direction of maximum linear velocity.



(a) The horizon for the 4-wheeled model



(b) The horizon for the 3-wheeled model

Figure 5 – Reachable horizon for $t_f \in \{0.05, 0.10, 0.15, 0.20, 0.25\}$s for two robot configurations, with optimal and naïve orientation-preserving controllers. Lines are broken where trajectory optimization was unable to find a solution.

Trajectory stabilization

Having found an optimal trajectory, it is useful to be able to stabilize it, to deal with disturbances, either in the form of model errors, or initial state errors.

One approach to this is to use TVLQR. We choose our cost matrices using Bryson's rule [2]:

$$Q = \begin{bmatrix} \ddots & & 0 \\ & \frac{\alpha_i^2}{(x_i)_{\max}^2} & \\ 0 & & \ddots \end{bmatrix}, \qquad R = \rho \begin{bmatrix} \ddots & & 0 \\ & \frac{\beta_i^2}{(u_i)_{\max}^2} & \\ 0 & & \ddots \end{bmatrix}$$

Where $\sum \alpha_i = 1,\ \sum \beta_i = 1$

In this case, we choose the maxima

$$(u_i)_{\max} = 1$$
$$\vec{x}_{\max} = [0.1\text{m}\ \ 0.1\text{m}\ \ \frac{\pi}{4}\text{rad}\ \ 1\text{ms}^{-1}\ \ 1\text{ms}^{-1}\ \ 3\pi\text{rads}^{-1}]$$

And the weights

$$\beta_i = 1, \qquad \vec{\alpha} \propto [10 \quad 10 \quad 100 \quad 1 \quad 1 \quad 1], \qquad \rho = \frac{1}{50}$$

Finally, we choose $Q_f = 100Q$.

Figure 6 shows the result of applying the resulting TVLQR controller to the optimal trajectory found in Figure 4, with perturbations in initial condition normally distributed as $N\left(0, \text{diag}\left(\frac{1}{2}\vec{x}_{\max}\right)^2\right)$.

TVLQR is not a perfect approach here, as it does not adjust the target based on the current position of the robot. Specifically, if the robot "falls behind" the target position, in our problem it is impossible for it to catch up, as the target path is the fastest of all possible paths.

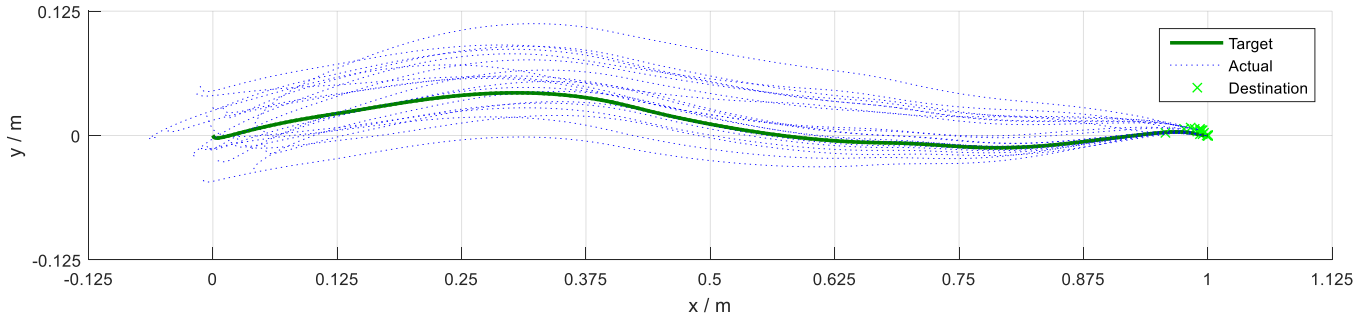


Figure 6 - Result of applying TVLQR to disturbed initial conditions. All final destinations are within 5% of the goal – so we still achieve better time-to-destination that if the naïve controller were used.

Conclusions

Using a more detailed model of the drive systems allows the dynamics to be exploited. However, the average improvement in travel time is small – around 20%. Additionally, trajectory optimization runs too slowly for this technique to be used online, in a real game.

Future work could try approximating the optimal controller with a closed-form controller, using offline trajectory optimization to tune its parameters.

References

[1] Russ Tedrake. Drake. [Online]. https://github.com/RobotLocomotion/drake

[2] MIT OpenCourseWare. (2010) 16.30 Lecture notes. [Online]. http://ocw.mit.edu/courses/aeronautics-and-astronautics/16-30-feedback-control-systems-fall-2010/lecture-notes/MIT16_30F10_lec12.pdf

[3] Walker Chan and et al. (2007) RoboCup Team Description Paper: RFC. [Online]. http://robocupssl.cpe.ku.ac.th/tdp/2007/RFCCambridge_SSL_TDP_2007.pdf

[4] Ricardo Cabello. Three.js. [Online]. http://threejs.org/