# Motion Planning using Waypoints

Randal W. Beard
Department of Electrical & Computer Engineering
Brigham Young University, Provo, Utah 84602

January 29, 2002

## 1 Introduction

The objective of these notes is to describe a deliberative approach to motion planning for mobile robots. In the deliberative approach, robot trajectories are planned explicitly. As a consequence timing can also be specified explicitly. The drawback of deliberative approaches is that they are strongly dependent upon the models used to describe the state of the world and to describe the motion of the ball and the robots. If the models are exact, then the motion planning techniques will be highly effective.

These notes are organized as follows. In Section 2 we discuss path planning using waypoints. In Section 3 we transform the waypoint path into a parameterized trajectory. In Section 4 we discuss control techniques for trajectory tracking using feedback linearization. One of the advantages of using waypoints is the ability to plan paths that intercept and shoot the ball. In Section 5 we describe ball prediction techniques. In Section 6 we discuss techniques for planning paths that intercept the ball.

## 2 Waypoint Planning

A waypoint path is sequence of straight-line path segments connecting points. An example of a waypoint path is shown in Figure 1.

A simple data structure can be used to store the waypoint path. One possibility is to store the waypoints, and an additional waypoint-pointer as part of the global state of the system. The waypoint-pointer indicates how

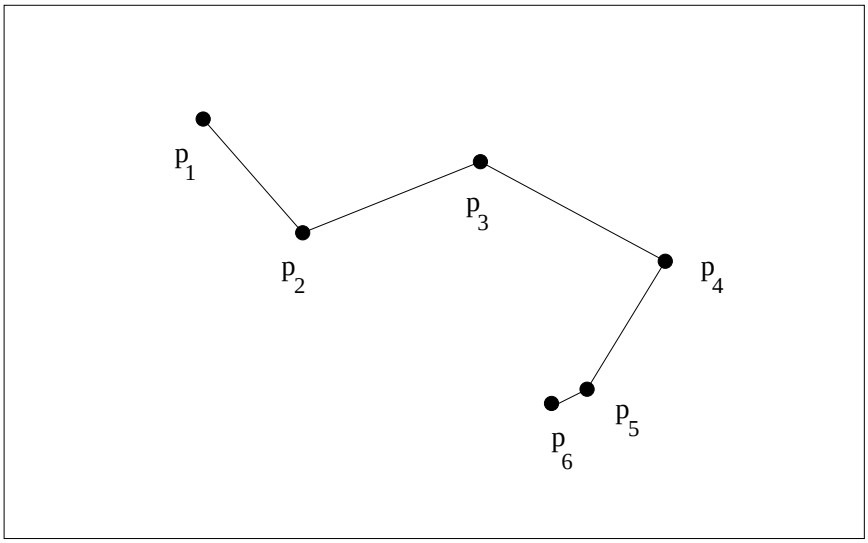


Figure 1: An example of a waypoint path.

many waypoint are current stored in the structure. An example of how to setup a waypoint data structure is shown in myrobot.h. Note that in addition the the waypoint-pointer `wpPtr` and the waypoint structure `wpPath`, there is also a waypoint path variable `wpTau`. This variable will be discussed in Section 3.

We suggest that you write several utilities that abstract the particular implementation of the waypoint structure. Possible utilities include

`utlWpClear`: Clear the waypoint stack.

`utlWpPush`: Push a waypoint on to the stack.

`utlWpPop`: Pop a waypoint from the stack.

`utlWpPrint`: Print the current waypoints for debugging.

## 2.1 Paths to Ball

As will be described in Section 3 we will not be able to cause the center of the robot to follow waypoint trajectories. As an alternative, we will control the "hand" position of the robot to follow the trajectory. The hand position

is defined as the point on the front of the robot that also lies on the line that is perpendicular to the wheel axes, intersecting the center of the robot, as shown in Figure 2.

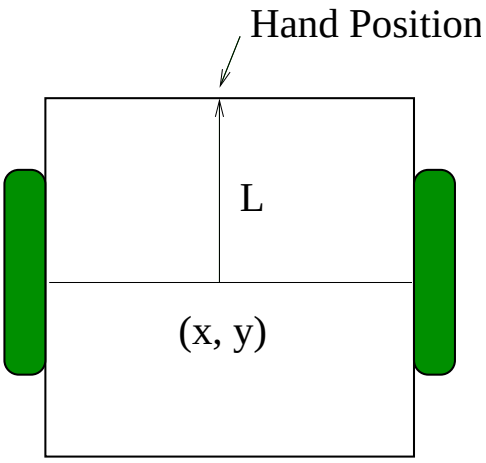


Figure 2: The hand position of the robot.

Since it is the hand position that will be following the trajectory, we must allow a little room for the robot to "straighten-out" after turns. Therefore, if we would like the robot to move from its current location and knock a stationary ball into the goal, we may plan a path like the one shown in Figure 3.

Suppose that the following variables are available to the path planning routine:

$$\begin{aligned} \mathbf{r} = (r_x, r_y)^T &\triangleq \text{ center position of robot,} \\ \psi &\triangleq \text{ heading angle of robot,} \\ \mathbf{b} = (b_x, b_y)^T &\triangleq \text{ ball position,} \\ \mathbf{g} = (g_x, g_y)^T &\triangleq \text{ goal position.} \end{aligned}$$

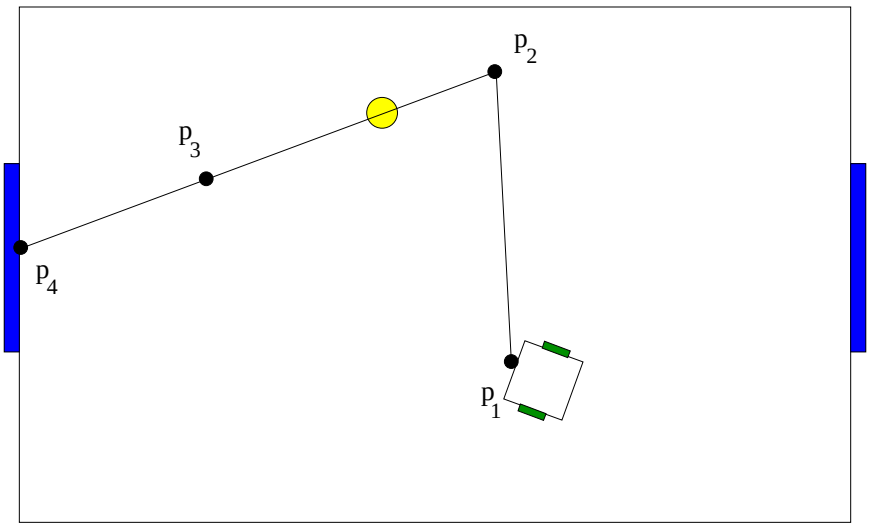


Figure 3: Waypoint path to the ball.

Then one possibility for the waypoints $\mathbf{p}_1$–$\mathbf{p}_5$ shown in Figure 3 are

$$\mathbf{p}_1 = L \begin{pmatrix} \cos(\psi) \\ \sin(\psi) \end{pmatrix}$$
$$\mathbf{p}_2 = \mathbf{b} - 2L \frac{\mathbf{g} - \mathbf{b}}{\|\mathbf{g} - \mathbf{b}\|}$$
$$\mathbf{p}_3 = \mathbf{b} + \frac{1}{2}(\mathbf{g} - \mathbf{b})$$
$$\mathbf{p}_4 = \mathbf{g}$$

## 2.2 Collision Avoidance

Consider the tasking of planning waypoint paths around an opponent as shown in Figure 4.

It is desirable to pick $\mathbf{p}_2$ such that the robot will not collide with the opponent as it moves toward the ball. Let $\mathbf{f}$ be the location of the opponent, and suppose that $R_f$ is its radius. Also assume that $R_r$ is the radius of our robot. Then we would like

$$\|\mathbf{p}_2 - \mathbf{f}\| \geq R_r + R_f.$$

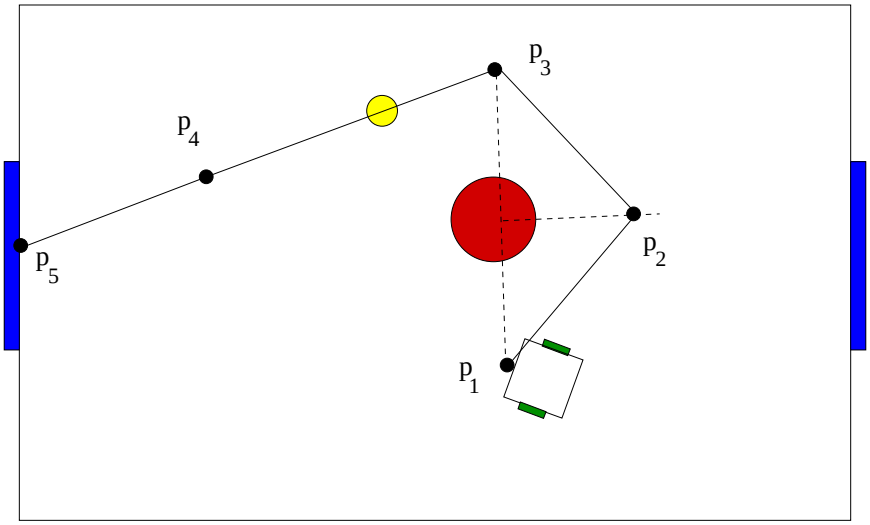


Figure 4: Waypoint path for Collision Avoidance.

The location of $\mathbf{p}_2$ will likely depend on which side of the $\mathbf{p_1}\bar{\mathbf{p}}_3$ line $\mathbf{f}$ is on, and how far $\mathbf{f}$ is from that line.

Consider Figure 5 which defines "RIGHT" and "LEFT" side of the line, as well as the angle $\theta$ and the distance $d$.

If the angle $\theta$ is positive, then we will say that $\mathbf{f}$ is to the LEFT of $\mathbf{p_1}\bar{\mathbf{p}}_2$. On the other hand if $\theta$ is negative than $\mathbf{f}$ is said to be to the RIGHT of $\mathbf{p_1}\bar{\mathbf{p}}_2$.

Recall that if $\mathbf{c} = \mathbf{a}\times\mathbf{b}$, then the signed magnitude of $\mathbf{c}$ is $c = \|a\|\,\|b\|\sin(\theta)$. If $\theta$ is positive then $c$ will be positive. If $\theta$ is negative then $c$ will be negative. In addition $|c|$ will be equal to the $d$ in the figure. Since

$$\begin{pmatrix} p_{2x}-p_{1x} \\ p_{2y}-p_{1y} \\ 0 \end{pmatrix} \times \begin{pmatrix} f_x-p_{1x} \\ f_y-p_{1y} \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ (p_{2x}-p_{1x})(f_y-p_{1y}) - (p_{2y}-p_{1y})(f_x-p_{1x}) \end{pmatrix},$$

if we let

$$c = (p_{2x}-p_{1x})(f_y-p_{1y}) - (p_{2y}-p_{1y})(f_x-p_{1x}),$$

then $d = |c|$ and

$$\text{sign}(c) = \begin{cases} 1 \implies \text{LEFT} \\ -1 \implies \text{RIGHT} \\ 0 \implies \textit{on line.} \end{cases}$$

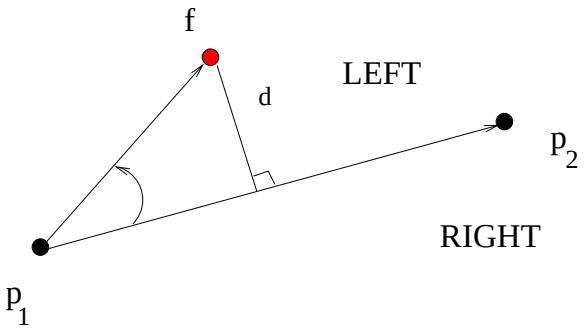


Figure 5: Determining the relative position of object with respect to waypoints $\mathbf{p}_1$ and $\mathbf{p}_2$.

The point $\mathbf{p}_2$ in Figure 4 will lie on the line that is perpendicular to the vector $\mathbf{p}_3 - \mathbf{p}_1$. Therefore, we need to know how to rotate a vector $\mathbf{q}$ by a vector $\theta$. Consider the geometry show in Figure 6.

Let $\mathbf{r}$ be the rotation of $\mathbf{q}$ by angle $\theta$ as shown in Figure 6. Also, let $\psi$ be the original angle of $\mathbf{q}$. Then, letting $\|\mathbf{r}\| = \|\mathbf{q}\| = \alpha$, we get

$$\begin{aligned} q_x &= \alpha \cos(\psi) \\ q_y &= \alpha \sin(\psi) \\ r_x &= \alpha \cos(\theta + \psi) \\ r_y &= \alpha \sin(\theta + \psi). \end{aligned}$$

Using the trig identities

$$\begin{aligned} \cos(\theta + \psi) &= \cos(\theta)\cos(\psi) - \sin(\theta)\sin(\psi) \\ \sin(\theta + \psi) &= \sin(\theta)\cos(\psi) + \cos(\theta)\sin(\psi), \end{aligned}$$

we get

$$\begin{pmatrix} r_x \\ r_y \end{pmatrix} = \begin{pmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{pmatrix} \begin{pmatrix} q_x \\ q_y \end{pmatrix}.$$

Letting

$$R(\theta) \triangleq \begin{pmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{pmatrix}, \tag{1}$$

then $\mathbf{r} = R(\theta)\mathbf{q}$, where $R(\theta)$ is called a rotation matrix.

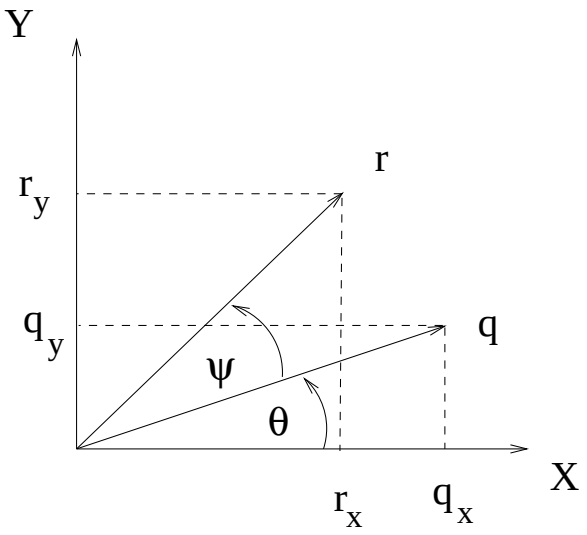


Figure 6: Coordinate frames for derivation or rotation matrix.

Therefore to place $\mathbf{p}_2$ to the LEFT of $\mathbf{f}$ as shown in Figure 4, we set

$$\mathbf{p}_2 = \mathbf{f} + (R_f + R_r)R(-\frac{\pi}{2})\frac{\mathbf{p}_3 - \mathbf{p}_1}{\|\mathbf{p}_3 - \mathbf{p}_1\|}.$$

# 3 Waypoint Trajectory Generation

In this section we will discuss techniques for transforming a waypoint trajectory into a trajectory. Let $\boldsymbol{\sigma}(\tau; \mathbf{p}_i, \mathbf{p}_{i+1})$ be the parameterized trajectory from $\mathbf{p}_i$ to $\mathbf{p}_{i+1}$, where $\tau \in [0, 1]$. Then

$$\boldsymbol{\sigma}(\tau; \mathbf{p}_i, \mathbf{p}_{i+1}) = (1 - \tau)\mathbf{p}_i + \tau\mathbf{p}_{i+1}.$$

Note that if $\tau = 0$ then $\boldsymbol{\sigma}(0) = \mathbf{p}_i$, and if $\tau = 1$, then $\boldsymbol{\sigma}(1) = \mathbf{p}_{i+1}$.

$\tau$ is a parameter that indicates the location of the desired trajectory between waypoints $\mathbf{p}_i$ and $\mathbf{p}_{i+1}$. The next step is to allow $\tau$ to vary with time. In other words we let $\tau = \tau(t)$. Therefore the desired trajectory is given by

$$\mathbf{z}^d(t) = \boldsymbol{\sigma}(\tau(t)).$$

The velocity of $\mathbf{z}^d$ is given by

$$\begin{aligned}\left\|\dot{\mathbf{z}}^d\right\| &= \|frac\partial\boldsymbol{\sigma}\partial\tau\|\,\dot{\tau} \\ &= \|\mathbf{p}_{i+1}-\mathbf{p}_i\|\,\dot{\tau}.\end{aligned}$$

Therefore, if $v^d$ is the desired velocity we should set

$$\dot{\tau} = \frac{v^d}{\|\mathbf{p}_{i+1}-\mathbf{p}_i\|}.$$

When the next waypoint $\mathbf{p}_{i+1}$ is reached, i.e., when $\tau = 1$, then $\tau$ will need to be reset to $\tau = 1$.

An example of a utility that computes a trajectory associated with two waypoints is given in trajGen.c.

# 4 Trajectory Tracking

In this section we describe how to use the feedback linearization technique to follow the trajectory generated in Section 3.

Consider the schematic of a mobile robot shown in Figure 2. Note that while the center of the robot cannot move in any direction, and is therefore not small signal controllable, the hand position, which we will call $\mathbf{z}$, can. The vector $\mathbf{z}$ is given by

$$\mathbf{z} = \begin{pmatrix} x \\ y \end{pmatrix} + L \begin{pmatrix} \cos(\psi) \\ \sin(\psi) \end{pmatrix}.$$

Differentiating $\mathbf{z}$ we get

$$\begin{aligned}\dot{\mathbf{z}} &= \begin{pmatrix} \dot{x} \\ \dot{y} \end{pmatrix} + L \begin{pmatrix} -\sin(\psi)\dot{\psi} \\ \cos(\psi)\dot{\psi} \end{pmatrix} \\ &= \begin{pmatrix} v\cos(\psi) - L\sin(\psi)\omega \\ v\sin(\psi) + L\cos(\psi)\omega \end{pmatrix} \\ &= \begin{pmatrix} \cos(\psi) & -L\sin(\psi) \\ \sin(\psi) & L\cos(\psi) \end{pmatrix} \begin{pmatrix} v \\ \omega \end{pmatrix}.\end{aligned}$$

We can therefore invert the nonlinearity by setting

$$\begin{aligned}\begin{pmatrix} v \\ \omega \end{pmatrix} &= \begin{pmatrix} \cos(\psi) & -L\sin(\psi) \\ \sin(\psi) & L\cos(\psi) \end{pmatrix}^{-1} \begin{pmatrix} u_1 \\ u_2 \end{pmatrix} \\ &= \begin{pmatrix} \cos(\psi) & \sin(\psi) \\ -\frac{1}{L}\sin(\psi) & \frac{1}{L}\cos(\psi) \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \end{pmatrix}.\end{aligned}$$

The feedback linearized system therefore becomes

$$z_x = u_1$$
$$z_y = u_2.$$

Given $z_x^d$ and $z_y^d$ we can design feedback control schemes according to the block diagram shown in figure 7.

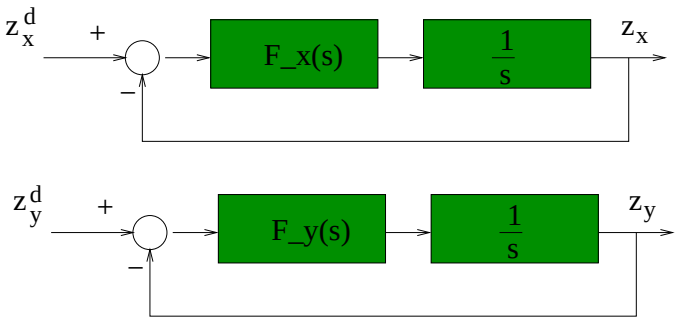


Figure 7: Feedback control loops for trajectory following.

Once $F_x(s)$ and $F_y(s)$ are designed, the trajectory following control systems can be implemented as shown in Figure 8.

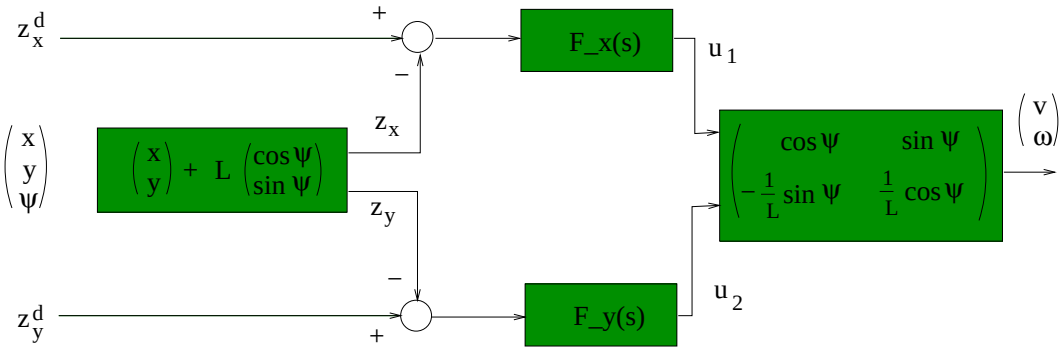


Figure 8: Control structure for trajectory following using feedback linearization.

# 5 Ball Prediction

The objective of this section is to describe a simple technique to predict future locations of the ball. When the ball is not in contact with the walls for another robot, then we will assume that there is no external force acting upon it (this assumes no air resistance or rolling friction). Under this assumption, by Newton's second law we have

$$m\ddot{\mathbf{b}} = 0,$$

where $m$ is the mass of the ball and $\mathbf{b}$ is its position vector. Dividing by $m$ and integrating once gives

$$\dot{\mathbf{b}}(t) = \dot{\mathbf{b}}(t_1),$$

where $t_1 \leq t$. Integrating again gives

$$\mathbf{b}(t) = \mathbf{b}(t_1) + \dot{\mathbf{b}}(t_1)(t - t_1).$$

Therefore if we measure the ball position at successive frames, then, in the absence of walls and obstacles, we could predict the ball position at for all future times as

$$\mathbf{b}(\tilde{t}) = \mathbf{b}[k] + \frac{\mathbf{b}[k] - \mathbf{b}[k-1]}{T}\tilde{t},$$

where $\tilde{t}$ is interpreted as future time from the current instance.

The ball may, however, collide with walls. We will assume perfectly elastic collisions, i.e., no energy loss. A property of elastic collisions is that the incidence angle is equal to the departure angle as shown in Figure 9.

We will derive the correction factor for the top wall, and leave the walls to the reader. Let YMAX denote the maximum Y coordinate of the soccer field. If $b_y(\tilde{t}) >$ YMAX, then the predicted position of the ball must be corrected to account for the interaction with the ball. Letting $d = b_y(\tilde{t}) -$ YMAX and assuming that the collision with the ball is elastic, the corrected predicted position of the ball is

$$\begin{aligned} b_y^+(\tilde{t}) &= b_y^-(\tilde{t}) - 2d \\ &= b_y^-(\tilde{t}) - 2b_y^-(\tilde{t}) + 2 \text{ YMAX} \\ &= 2 \text{ YMAX} - b_y^-(\tilde{t}), \end{aligned}$$

where $b_y^-(\tilde{t})$ denotes the predicted $y$-position of the ball before correcting for the collision, and $b_y^+(\tilde{t})$ denotes the predicted $y$-position of the ball after correcting for the collision.

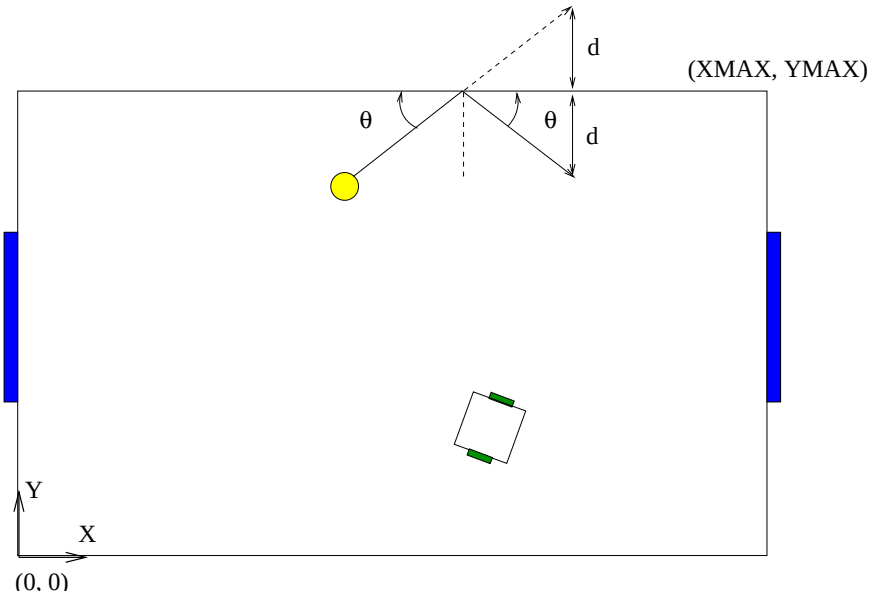


Figure 9: Elastic collision.

An example of c-code that performs ball prediction, correcting for collisions with the wall is ballPredict.c

You may also want to take into account robot collisions. In particular it would be nice to know how your robot will effect the ball. As a precursor to robot collisions, lets rework the wall collision equations, putting them into a more general form. Let $\dot{\mathbf{b}}^-$ be the velocity of the ball before the collision, and let $\dot{\mathbf{b}}^+$ be the velocity of the ball after the collision. Since the collision is elastic, the magnitude of $\dot{\mathbf{b}}^+$ equals the magnitude of $\dot{\mathbf{b}}^-$, however the direction has changed by an angle of $2\theta$. Therefore

$$\dot{\mathbf{b}}^+ = R(2\theta)\dot{\mathbf{b}}^-,$$

where $R(\cdot)$ is the rotation matrix given in Equation (1). Therefore, the ball prediction equation after one bounce becomes

$$\mathbf{b}(\tilde{t}) = \mathbf{b}[k] + t_{coll}\dot{\mathbf{b}}[k] + (\tilde{t} - t_{coll})R(2\theta)\dot{\mathbf{b}}[k],$$

where $t_{coll}$ is the predicted time of collision. These equations can be extended to multiple bounces in a straightforward manner.

Now suppose that the ball collides with a robot that has velocity vector $\mathbf{v}$ as shown in Figure 10. The collision imparts velocity to the ball. Since the ball dynamics are linear, by superposition the new velocity of the ball will be

$$\dot{\mathbf{b}}^+ = R(2\theta)\dot{\mathbf{b}}^- + \mathbf{v},$$

where $\theta$ is the angle of incidence of collision with the face of the robot.

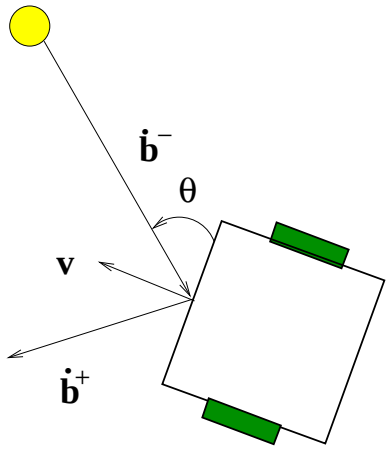


Figure 10: Ball-robot collision

# 6 Ball Intercept

In this last section we address the problem of planning waypoint trajectories that intercept the ball along a specified vector $\mathbf{h}$ at a robot speed $v$ as shown in Figure 11. The intercept time will be denotes as $T$.

We will assume that the following functions are available

$\tilde{b} = \mathbf{ballPredict}(\tilde{t})$**:** Returns the predicted ball location at time $\tilde{t}$ into the future.

$\mathcal{P} = \mathbf{planPath}(\mathbf{p}_1, \mathbf{p}_2, \mathbf{h})$**:** Plans a waypoint path from point $\mathbf{p}_1$, to point $\mathbf{p}_2$, such that the direction of the final leg of the path is identical to $\mathbf{h}$. We assume that collision avoidance is implemented as part of this function.

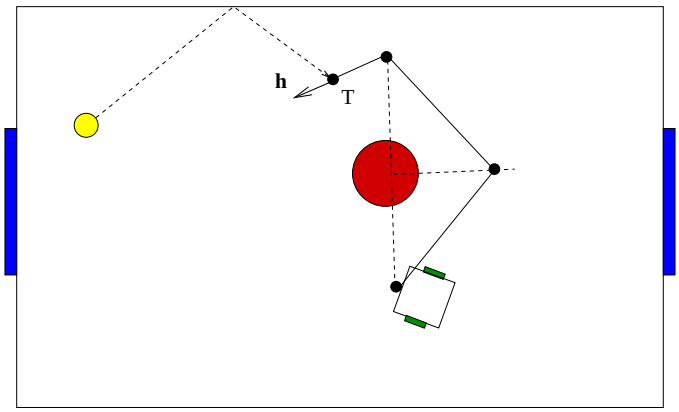


Figure 11: Robot intercepting a ball along vector $\mathbf{h}$.

$L = \textbf{pathLength}(\mathcal{P})$**:** Returns the path length of $\mathcal{P}$.

For any $\tilde{t} \geq 0$ we can compute the path length to the predicted ball location as

$$L = \text{pathLength}(\text{planPath}(\mathbf{z}[k], \text{ballPredict}(\tilde{t}), \mathbf{h})).$$

The time it takes the robot to traverse this path at speed $v$ is given by

$$T(\tilde{t}) = \frac{\text{pathLength}(\text{planPath}(\mathbf{z}[k], \text{ballPredict}(\tilde{t}), \mathbf{h}))}{v}.$$

Define the function

$$g(T) = vT - \text{pathLength}(\text{planPath}(\mathbf{z}[k], \text{ballPredict}(T), \mathbf{h}))$$

An *intercept time* is defined to be any $T^*$ that satisfies the equation

$$g(T^*) = 0.$$

The objective is to find the minimum intercept time, and to follow the resulting path. The minimum intercept time can be found via a binary, or bisection, search algorithm. Note that $g(0) < 0$, and that for some $T > 0$, we are guaranteed that $g(T) > 0$. Therefore the following algorithm will quickly converge to $T^*$.

**Bisection Search**

**Step 1.** Set $T = 0$, $T_1 = 0$.

**Step 2.** Incrementally increase $T$ until $g(T) > 0$. Set $T_2 = T$.

**Step 3.** While $\left|g\left(\frac{T_1+T_2}{2}\right)\right| > \epsilon$ if $g\left(\frac{T_1+T_2}{2}\right) > 0$ set $T_2 = \frac{T_1+T_2}{2}$, else set $T_1 = \frac{T_1+T_2}{2}$.

**Step 4.** Set $T^* = \frac{T_1+T_2}{2}$.

Once $T^*$ is found, plan a path from $\mathbf{z}[k]$ to ballPredict($T^*$), to solve the intercept problem