15-424 Lab 6: Safe Ball Passing in RoboCup

Alexei Colin

November 19, 2014

1 Introduction

Robotic soccer, introduced via the RoboCup competition, presents a rich controller design challenge. In Small Size League a team of six small round robots with omni-directional wheels and ball-kicking solenoids is controlled via a central program (*Mission Control*), with global knowledge about the field, the players and the ball. The ball is a standard golf ball. A kick-off is pictured in Figure 1.

The rule book for robotic soccer is not much different from that for human soccer. To avoid penalty cards and damage, the team of robots must avoid physical contact with each other, with the opponents, and with the walls. To stand a chance for victory, the team should keep the ball within the field boundaries. The basic game-play strategy is passing the ball to a team-mate that is closer to the goal.

In this work, we will focus only on one team's robots (no opponents) and on a common gameplay strategy of passing the ball without letting it go out of bounds.

We verified the safety property for a hybrid program for a team of exatly two robots. However, the hybrid program was designed so that adding more robots amounts to no more changes than mechanically duplicating expressions. Not nearly as general as an HP in quantified differential dynamic logic would be, but designed with extension to multiple teammates in mind. The source and destination notions described in Section 2.4 make this possible. With two robots, neither can be an obstacle, so some branches of the proof that are trivial would no longer be trivial with more than two robots.[1]

[1]Alas, the proof attempt for a three-robot team, did not succeed before time ran out, however hope that the model does readily generalize lives on.

Figure 1: Kickoff in RoboCup Small Size League.

2 Model

2.1 Field

The field and all its contents is modeled in two dimensions, with the origin at the center of the field. The field is a rectangle of size $F \times F$. On the real RoboCup field, robots can choose to chip-kick the ball in a third dimension *over* other robots. A 3D model would be essential to cover the game fully, however the sacrificed dimension wins tractability at a cost that is much less than "33% of the game".

2.2 Ball

The ball is modeled as a point, $(B_{\mathrm{x}}, B_{\mathrm{y}})$. The ball moves in straight lines with constant speed B_{s} in any direction denoted by a normalized vector $(B_{\mathrm{dx}}, B_{\mathrm{dy}})$.

A point-mass model is simple yet powerful enough to subsume most of the effects that a size-aware model would add, such as spin, since these effects can be described at the trajectory level and since the ball is very small compared to the robots (golf ball). We expect that even most advanced models of the game would model the ball as a point-mass. We simplify further by neglecting the mass, thus letting the ball move at constant speed, and leave non-zero mass as an extension.

Upon hitting a robot, the ball is either captured by the robot's ball handling mechanism (a dribbler) or reflected. Which of the two outcomes happens depends on where the ball comes in contact with the robot. The conditions under which the robot's dribbler can capture the ball and the direction of the reflection is defined by the robot model.

2.3 Robots

Small-Size League robots are small cylinders, usually with a flat front as is visible in Figure 1. Real robots are equipped with omni-wheels arranged in a diamond pattern, which lets them spontaneously move in any direction. However, in this work we focus on controlling the ball kicking and work with stationary robots. Each robot is equipped with a radio, which acts as the ultimate "sensor" for positions and velocity vectors of all other robots and the ball (broadcast by Mission Control). Global knowledge is a fundamental aspect of the Small-Sized League. Hence, the controllers we model make use of the global state.

We next introduce key aspects of the robot that any model must define. Then, we describe two models that were considered in this work and motivate the ultimate choice.

Each robot has a ball handling mechanism, which is usually a dribbler. A dribbler is a short rubberized shaft that rotates towards the inside of robot. Should the ball come in contact with it at a suitably low velocity, the dribbler would put a spin on the ball towards the robot, and keep the ball in place even as the robot moves. For a pass to be received successfully, the ball must reach the robot within the boundaries of this mechanism. If the ball reaches the robot at some other point, then it is reflected from the robot, as a golf ball would be upon hitting a hard surface. Ball reflection off of robots is very common in actual RoboCup matches and is most often due to either miss-aimed kicks or by robots ending up on the ball trajectory either deliberately for defense or inadvertently.

We model the ball contact location requirement for successful pass receipt and the ball reflection behavior. The direction of reflection is defined by the each robot model. Recall from our simplified ball model that the ball moves at a constant velocity. If any passes are to be successful, we must assume that the fixed ball velocity is low enough for a dribbler to capture the ball. This eliminates the need to explicitly model the maximum ball velocity for pass receipt.

Two models were considered: a segment model and a disk model. The segment model was chosen in favor of the simpler occluded region calculation However, the disk model is a more faithful and more flexible representation. The occluded region calculation for the disk model is included to serve as a basis for extending this work. The evolution domain of the ball is limited to the non-occluded region.

2.3.1 Segment Model

A robot is represented as a one-dimensional vertical segment of length R_r as shown in Figure 2. The position of the robot is represented by the center of the segment $(R_{i,\mathrm{x}}, R_{i,\mathrm{y}})$. The robot's ball handling mechanism (dribbler) is located at the center of the segment. In order for the dribbler to capture the ball, the ball must come in contact with the robot at exactly the dribbler location, i.e. at $(R_{i,\mathrm{x}}, R_{i,\mathrm{y}})$. From all other contact points, the ball is reflected with respect to the horizontal axis as shown in Figure 3. The reflection dynamics are described more formally in Section **??**.

While this model appears to be far from reality, it retains the essence of the robot's effect on the ball. At commonly sharp angles of passes and shots, the segment obstructs the line-of-sight in a similar way as a disk would. Also, a collision with and reflection of the ball from a vertical "wall" is a closer model for the flat-front robots than a reflection from a perfect disk. The orientation of the robot segments is chosen to be perpendicular to the orientation of the goal, since we target a gameplay of "zig-zag" passing towards the goal, in which passes are roughly parallel to the goal. We prevent infinities by disallowing vertical alignment among robots, and hence vertical passes.

Zero-Area Property and Occluded Region:

In the segment model the robots have zero area. This characteristic is a two-edge sword. On one hand, it simplifies the ball dribbler mechanism modeling to the bare minimum and, consequently, does not require the robot orientation to be modeled. The segment-robot can receive a pass from any direction except along a vertical trajectory. There is no special handling of the vertical in the robot model; instead, such are denied by restricting robot positions to disjoint x-coordinates. On the other hand, the zero-area characteristic complicates the collision detection logic for a somewhat subtle reason as outlined below.

To detect collision between the ball and any segment robot is not as straightforward as checking that the ball is not within any robot segment, because the robot segment needs to be a boundary of the evolution domain. The evolution domain must include its boundary, because otherwise continuity is lost between the cyber and the physical. When the physics evolution ends, it always ends within the domain. If the robot segment is outside the domain, then the cyber part that runs after the physics will never see the case when the ball is at a robot, i.e. is located at the robot's ball handling mechanism, i.e. the center of the robot segment. However, if the robot segment is included into the evolution domain, there is noting stopping the physics from taking the ball on a trajectory straight through a robot without any interruption.

To work around this consequence, we let the evolutionary domain include the robot segment, but exclude an *occluded region* "behind" the robot, where "behind" is defined relative to the direction of the ball x-velocity. The non-occluded region is the area through which the ball can physically travel in a straight trajectory without having collided with the segment robot. The complement of this region is the occluded region. In Figure 2 the occluded region is shaded.

2.3.2 Disk Model

The disk model is a higher-fidelity alternative to the segment model. A robot is represented as a perfect disk, neglecting the frontal flatness, centered at the robot position (R_x, R_y). The coordinate system is translated such that the origin is at the ball position (B_x, B_y) in order to facilitate the derivation of the expression for the occluded region boundary.

Since a disk has a non-zero area, the disk model does not require to compute the *occlusion region* needed for a zero-area segment robot model as discussed in Section **??**. In the non-zero area model, the collision check would be a simple ball-within-robot check. There would be no issues with including the boundary of the disk into the evolution domain. However, the disk model requires a more complicated notion of the ball handling mechanism in the robot.

There must be a way for the ball to make contact with the robot that is not classified as a collision. Letting the robot receive passes (only) at its center, as we do in the segment model, would no longer work, since the collision condition would trigger before the ball would reach the robot center. Therefore, the ball handling mechanism would need to be on the edge of the robot and the kicking robot would need to direct its kick towards the location of the mechanism. This seems doable, however this leads to the robot acquiring an orientation. The robot would only be able to receive passes from a certain side: the side that is reachable by the ball trajectory without passing through the robot. We could let the robot receive passes from all directions by endowing it with two ball handling mechanisms: one on each side. However, the kicking robot would still need to be aware of this and explicitly figure out which mechanism to aim its pass for. Furthermore, there remains the question of the flat front, particularly relevant for ball reflection dynamics.

The complexity in ball handling mechanism that in a non-zero area disk model was weighed against the complexity in collision check in a zero-area segment model. The former appeared less formidable and was undertaken. Retrospectively, the disk model should be the choice going forward, not only due to its realism, but also because even if it is more complex, it is less hacky, which should lead to a more tractable hybrid program.

2.4 Source and Destination

Our controller, physics, and proof all rely on keeping track of the *source*, S, and *destination*, D of the ball. These are defined whenever the ball is in mo-

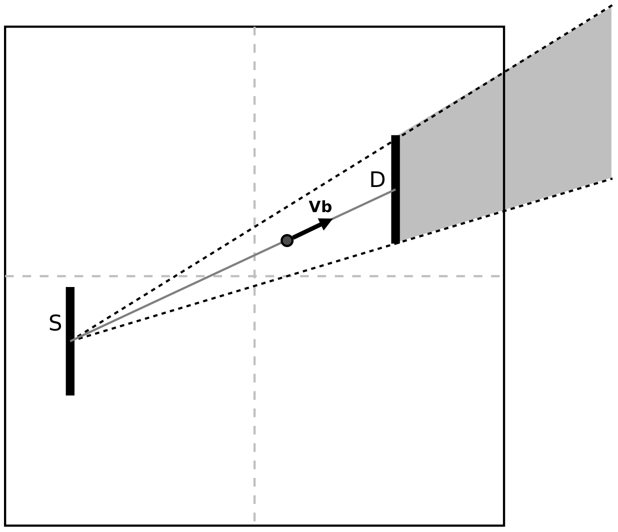

Figure 2: A successful ball pass in progress. Evolution domain excludes the shaded region.

tion. Only when the ball is stationary neither of these is necessary. Whenever the direction of the ball is set in the hybrid program, the source and destination are updated accordingly. In case of reflection, the destination *should* be set to the nearest intersection of the direction vector with a robot (obstacle) or the field boundary.

An assumption implicit in our hybrid program is that an unobstructed teammate always exists. The HP aborts should this not be the case, which is not acceptable if a controller is to be verified in a world where this assumption does not hold. There is little excuse for not adding a non-deterministic choice to keep the ball in the possession of the robot that has it.[2]

2.5 Dynamics

A robot influences the motion of the ball in two cases. Upon a successful pass, the receiver takes possession of the ball and can redirect it as it sees fit. Figure 2 shows an evolution corresponding to a successful pass in progress. The shaded occluded region excluded by the evolution domain formula. Note that this region is a function of which robot is kicking and which receiving.

[2]An attempt was made to generalize the HP such that the chosen *destination* for the kick can be the same as the *source*, however success was not achieved before time ran out.

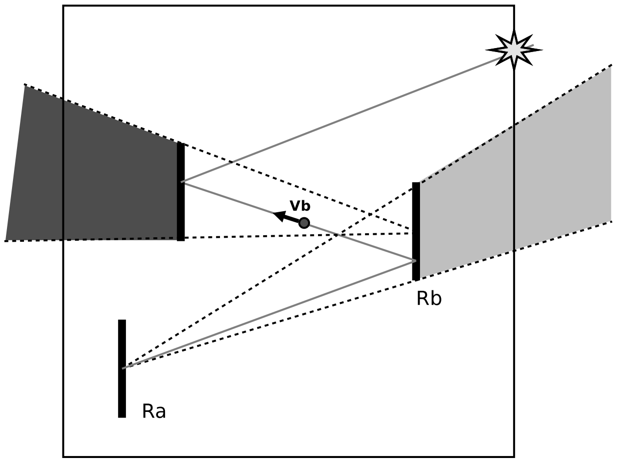

Figure 3: Ball reflection which results from an unsuccessful pass. Three sequential evolutions are shown.

Upon an unsuccessful pass, or an arbitrary collision with the ball, the ball reflects off of the robot as it would reflect from a flat vertical wall. Nothing else affects the motion of the ball, which implies the ball would roll out of bounds if its path is clear. Figure 3 illustrates this in a game-play example of three consecutive evolutions. The ball is kicked by R_a but misses R_b receptacle and is reflected twice before going out of bounds.

A successful path is one in which the ball reaches exactly the midpoint of the robot-segment, where the ball capturing and kicking mechanism is located. Upon contact at any other point on the robot the ball is reflected. An (very feasible) extension for flexibility and realism is expanding the acceptable receiving point to a small range around the midpoint. On successful repossession, the receiver can send the ball wherever by directly setting its direction vector. We do not restrict the outgoing angle and let the ball be launched from either side of the robot-segment.

This dynamics is implemented as the "physics" part of the hybrid program, which spans both the continuous differential equation with its evolution domain and some discrete logic. The collision event is detected by including into the evolution domain only the cone region which the ball can reach from the kicker without collision, which is calculated as above and below the two rays from the kicking robot to the extremities of each of the other robots (obstacles). That is,

$$B_{\mathrm{y}} \leq \frac{(R_{2,\mathrm{y}} - R_{\mathrm{r}}) - S_{\mathrm{y}}}{R_{2,\mathrm{x}} - S_{\mathrm{x}}}(B_{\mathrm{x}} - S_{\mathrm{x}}) + S_{\mathrm{y}}$$
$$B_{\mathrm{y}} \geq \frac{(R_{2,\mathrm{y}} + R_{\mathrm{r}}) - S_{\mathrm{y}}}{R_{2,\mathrm{x}} - S_{\mathrm{x}}}(B_{\mathrm{x}} - S_{\mathrm{x}}) + S_{\mathrm{y}}$$

An event-triggered hybrid program is a fair approximation to the robot behavior because the ball receiving mechanism traps the ball without any action by the microcontroller that is limited to a finite frequency. In a more faithful time-triggered model would be a delay (due to finite frequency) between receiving the ball and kicking it again. During this delay the ball would remain stationary inside the robot's receptacle, so it is of little interest to model it.

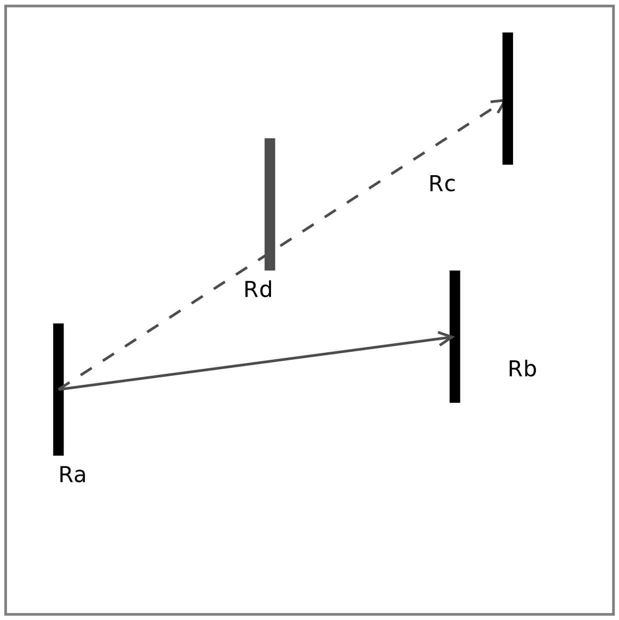

Figure 4: Controller needs to choose a non-occluded pass receiver.

3 Purpose and Value

The robot controller is responsible for choosing the teammate to which to pass the ball and steering the ball towards it. Should it make a bad choice, e.g. R_c instead of R_b in Figure 4, or send the ball in a wrong direction, the ball will likely go out of bounds and all hope for victory would be lost. We proved that this cannot happen with our basic controller that sends the ball to a randomly chosen *non-occluded* teammate by calculating the direction vector from the source and destination points. Both the choice and the calculation are correct.

As a result of nondeterministic choice of the destination, this controller is flexible enough to include any logic for choosing the recipient of the pass as long as the given occlusion checks are in place. An examples of a controller to which the proof still applies is one that chooses the teammate closest to the goal among unoccluded ones.

The present controller might stand out as very simple in light of the effort required to verify but one property of it. However, a controller similar in spirit but extended in functionality, for example, one that targets a world where robots could move, might wish to calculate ball trajectories that are safe from interception. A bet that an off-hand design of this moderately more advanced controller would be correct for any robot and ball positions possible on the field does not feel quite like a sure winner. A formal proof of safety similar to the one done in this work would bring confidence into the design.

4 Proof Techniques

The proof is built from these three key pieces:

1. The ball will move along the source-destination line

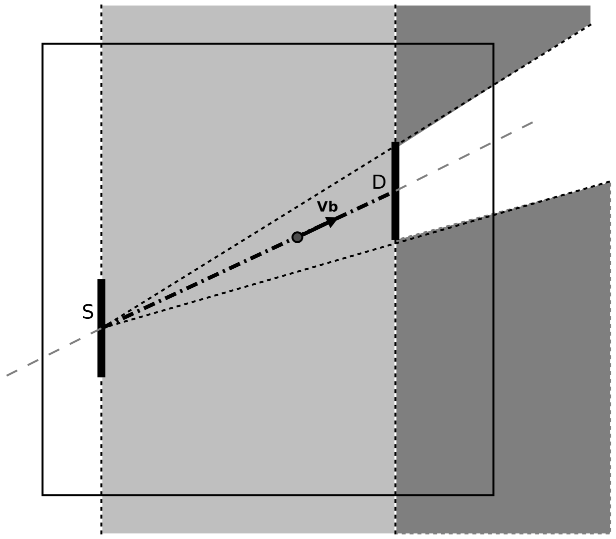

Figure 5: The "intersection" of invariants constrains the motion of the ball to be along the bold dash-dotted line.

2. The ball will start from the "front" of the source
3. The ball will stop moving when it hits the destination.

Together these imply that the ball will move between the source and the destination, which will keep it within bounds as long as the former are. The region delineated by the invariants is the bold dash-dotted line in Figure 5.

4.1 Differential and Loop Invariants

The differential invariant gives the first of the above three pieces. It states that the robot always moves along the line (not segment) between the source, S, i.e. the kicker or reflecting point, and the destination, D, i.e. the receiver or a dummy destination. That is,

$$\phi \equiv (B_\mathrm{y} - S_\mathrm{y})(D_\mathrm{x} - S_\mathrm{x}) = (D_\mathrm{y} - S_\mathrm{y})(B_\mathrm{x} - S_\mathrm{x})$$

The loop invariant makes several strong statements:

- En-route from robot to robot: the source and the destination are always two (distinct) robots
- Moves on a line from source to destination or is stationary (the differential invariant condition)
- Moves within the segment between source and destination (stronger than the differential invariant, relies on the initial conditions at which the differential equations start)
- No collisions: the ball can never be in the position about to be reflected, i.e. it is either exactly in the middle of any robot segment or outside it.
- Ball is within field boundaries

5 Conclusion and Future Work

We have developed a basic hybrid program for modeling ball passing among robotic football players in RoboCup Small Sized League. A proof that a simple controller keeps the ball within field boundaries was derived.

This work asks to be extended in two directions: to make the model more realistic and to prove other safety and efficiency goals. Among the most plausible extensions are

- an ability to kick the ball into a goal[3]
- disk robot model with an orientation and a dribbler that is sensitive to incoming velocity and angle of the ball
- ability of robots to move (while ball stationary)
- a ball with non-zero mass subject to friction

[3]Attempted at length, but no proof. In another attempt, instead of letting the ball evolve outside the field if it went through the goal, I would take a different approach: at the goal, stop the physics just before it reaches the boundary ($B_y < F$), but anywhere else on the boundary, continue the physics ($B_y \leq F$), and have the safety condition be the strict inequality.

A safety goal of interest is to ensure the ball stays above a certain velocity whenever it moves, which minimizes the possibility of interception by the opponent. A controller that could do this would need to pick partners that are close and would not shoot at the goal from far away. An efficiency goal could be to ensure there is a way for the ball to get to the goal.

The above list of extensions is far from exhaustive, however, at this point it as clear as ever how simple extensions can quickly make the hybrid program or the proof enormously more difficult.

6 Appendix: Robot Controller Hybrid Program

```
/* Lab6: safe ball passing in Robocup Small-Size-League */

\functions {
   R F; /* field size (FxF) */
   R Rr; /* robot radius */
   R Bsk; /* constant ball speed */

   \external R Sqrt(R);
}

\programVariables {
   /* Robots */
   R R1x; R R1y;
   R R2x; R R2y;

   /* The ball */
   R Bx; R By; /* ball position */
   R Bs; /* ball speed (equals either Bsk or zero) */
   R Bdx; R Bdy; /* ball direction (a normalized vector) */

   /* Intermediate helper variables */
   R Sx; R Sy; /* Source: position of the robot kicking the ball */
   R Dx; R Dy; /* Destination: target position to where the ball is kicked */
   R rx; R ry; /* Vector from kicker to receiver */
   R norm_r; /* length of the kick vector */
   R ROx; R ROy; /* potential obstacle robot position */
   R ROiy; /* intersection of obscle robot with line of sight */

   R t; /* evolution time */

   /* Initial value ghosts */
   R Bs0;
   R Bx0; R By0;
   R Bdx0; R Bdy0;
}

\problem{
(
/* Parameters are sane */
F > 0 & Bsk > 0 & Rr > 0 &

/* Robots are inside the field */
-F < R1x - Rr & R1x + Rr < F & -F < R1y - Rr & R1y + Rr < F &
-F < R2x - Rr & R2x + Rr < F & -F < R2y - Rr & R2y + Rr < F &

/* Robots do not overlap in the x dimension (no vertical passing) */
(R1x - Rr > R2x + Rr | R2x - Rr > R1x + Rr) &

/* Ball is inside the field */
-F < Bx & Bx < F & -F < By & By < F &

/* Ball is stationary */
Bs = 0 & Bdx = 0 & Bdy = 0

)
```

```
->
\[
   /* Initialize helper variables to satisfy the loop invariant */
   Sx := R1x;
   Sy := R1y;
   Dx := R2x;
   Dy := R2y;

   (
    /* If a robot has the ball... */
    if (Bx = R1x & By = R1y |
        Bx = R2x & By = R2y)
    then

      /* ... then, the robot chooses a teammate and passes the ball to it */

      /* This sub-program models the controller decision procedure that
       * chooses the teammate to whom to pass the ball.
       *
       * The choice of teammate is modeled as a nondeterministic choice
       * among the non-occluded teammates. The HP models this by first
       * nondeterministically choosing any destination, and then aborting
       * paths where the chosen destination is occluded.
       *
       * Note: an unfortunate assumption is that a non-occluded teammate
       * always exists. There's no excuse for not adding a choice to keep
       * the ball, however a proof atempt for a modified HP did not
       * succeed before time ran out.
       */

      /* source location of the kick */
      Sx := Bx; Sy := By;

      /* destination of the kick */
      (Dx := R1x; Dy := R1y)
      ++
      (Dx := R2x; Dy := R2y);

      /* Don't pick ourselves as the kick destination */
      ?(Sx != Dx | Sy != Dy);

      /* Check if any of the other robots obstruct the pass trajectory
       * between source and dest; and abort any controller code paths
       * which picked an obstructed destination. This is done by
       * checking that the pass trajectory does not intersect any of
       * the robot segments.
       *
       * The following steps are repeated for each robot RO.
       *
       * Find point (ROx, ROiy): the intersection of ball trajectory with the
       * line that contains the obstacle robot segment ('i' for
       * 'intersection'). The ball trajectory is the line from source (Sx, Sy)
       * to destination (Dx, Dy). The line that contains the obstacle robot
       * segment is a vertical line through ROx.
       *
       *     ROiy := *; (?((ROiy-Sy)*(Dx-Sx) = (Dy-Sy)*(ROx-Sx)));
       *
   * The robot does not obstruct a pass if either:
   *   (A) the robot is the source (ourselves) or the destination,
   *   (B) the ball trajectory does not pass through the robot
   *       segment, i.e. the intersection (R0x, R0cy) lies
   *       outside of the robot segment.
   *
   *      ?(((R0x = Sx & R0y = Sy) | (R0x = Dx | R0y = Dy)) |
   *       (R0iy > R0y + Rr | R0iy < R0y - Rr));
   */

  /* Robot 1 */
  R0x := R1x; R0y := R1y;
  R0iy := *; (?((R0iy-Sy)*(Dx-Sx) = (Dy-Sy)*(R0iy-Sx)));
  /* It's not an obstacle if it is ourselves or the destination */
  /* Is in the line of sight */
  ?(((R0x = Sx & R0y = Sy) | (R0x = Dx | R0y = Dy)) |
    (R0iy > R0y + Rr | R0iy < R0y - Rr));

  /* Robot 2 */
  R0x := R2x; R0y := R2y;
  R0iy := *; (?((R0iy-Sy)*(Dx-Sx) = (Dy-Sy)*(R0iy-Sx)));
  /* It's not an obstacle if it is ourselves or the destination */
  /* Is in the line of sight */
  ?(((R0x = Sx & R0y = Sy) | (R0x = Dx | R0y = Dy)) |
    (R0iy > R0y + Rr | R0iy < R0y - Rr));

  /* The kick vector: Vector from source to the destination */
  rx := Dx - Sx;
  ry := Dy - Sy;

  /* Pass distance, for getting a normalized ball direction vector */
  norm_r := *; ?(norm_r^2 = rx^2 + ry^2 & norm_r > 0);

  /* Set the direction and speed of the ball */
  /* SMT does not like division
  Bdx := rx/norm_r;
  Bdy := ry/norm_r;
  */
  Bdx := *; ?(Bdx*norm_r = rx);
  Bdy := *; ?(Bdy*norm_r = ry);
  Bs := Bsk

else if ( /* Ball reflection physics */
         /* If ball is moving... */
         ((Bs*Bdx)^2 + (Bs*Bdy)^2 > 0) &
         /* ...and has hit a robot outside of the robot's
            'ball receiver' mechanism */
         (Bx = R1x & R1y - Rr < By & By < R1y + Rr |
          Bx = R2x & R2y - Rr < By & By < R2y + Rr))
     then
        /* Set the impact position as the source of the "kick" */
        Sx := Bx;
        Sy := By;

        /* Reflect the ball */
        Bdx := -Bdx;
        Bs := Bsk;
        /* Let the destination of the emulated "kick" be a waypoint ahead
         * of the ball position in the direction of the reflection. This
         * satisfies the evolution invariant that the ball must move in a
         * line between source and destination.  Note that destination (D)
         * is not used in the evolution domain; it is used only in the
         * invariants and as a local (within-iteration) variable in the
         * cyber controller. It does not affect physics at all, it is a
         * 'tool' that we use for the proof. If reflection does happen, the
         * ball will keep rolling until it goes out of bounds or hits
         * another robot (and reflects again, etc.) regardless of what we
         * set the destination to here.
         */
        Dx := Bx + Bdx;
        Dy := By + Bdy
     fi
fi;

/* Next comes the physics */

/* Ghosts */
Bdx0 := Bdx;
Bdy0 := Bdy;
Bs0 := Bs;
Bx0 := Bx;
By0 := By;

t := 0;
(
 /* Evolve: let the ball loose!
  * The ball evolution is split into two cases (two choices):
  *     (A) the ball is in motion (positive speed)
  *     (B) the ball is stationary
  * The differential equations are the same, however the we cannot maintain
  * the same invariant (that the ball is on a line between source and
  * destination) for a stationary ball since the initial position of the
  * ball is unrestricted (as long as it is stationary).
  */

 /* Ball evolution: case (A): ball in motion */
 {Bx' = Bs*Bdx, By' = Bs*Bdy, Bdx' = 0, Bdy' = 0, t' = 1,
  /* Anything goes as long as the ball is stationary */
  (Bs*Bdx)^2 + (Bs*Bdy)^2 > 0 &

   /* otherwise, physics could take the ball only into this region: */
   (
    /* ... "in front of" the kicking (or reflecting) robot */
    (Bdx > 0 & Bx >= Sx | Bdx < 0 & Bx <= Sx) &

    /* detect collission with a robot */

    (
      /* Robot 1 */
      (
        /* region in front of the collission candidate robot */
        (((Bdx > 0 & Bx <= R1x) | (Bdx < 0 & Bx >= R1x)) | /* union with */
            /* ... ... or "behind" the destination robot, but
                        not having passed through it (i.e. not in the "shadow") */
            /*
            (By >= ((R1y+Rr-Sy)/(R1x-Sx))*(Bx-Sx) + Sy &
             By <= ((R1y-Rr-Sy)/(R1x-Sx))*(Bx-Sx) + Sy)
            */
             (
               ((By - Sy)*(R1x-Sx) >= (R1y+Rr-Sy)*(Bx-Sx) & R1x - Sx >= 0 |
                (By - Sy)*(R1x-Sx) <= (R1y+Rr-Sy)*(Bx-Sx) & R1x - Sx < 0) &
               ((By - Sy)*(R1x-Sx) <= (R1y-Rr-Sy)*(Bx-Sx) & R1x - Sx >= 0 |
                (By - Sy)*(R1x-Sx) >= (R1y-Rr-Sy)*(Bx-Sx) & R1x - Sx < 0)
             )
            )
          ) &
          /* Robot 2 */
          (
            /* region in front of the collission candidate robot */
            (((Bdx > 0 & Bx <= R2x) | (Bdx < 0 & Bx >= R2x)) |

            /* ... ... or "behind" the destination robot, but
                        not having passed through it (i.e. not in the "shadow") */
            /*
            (By >= ((R2y+Rr-Sy)/(R2x-Sx))*(Bx-Sx) + Sy &
             By <= ((R2y-Rr-Sy)/(R2x-Sx))*(Bx-Sx) + Sy)
            */
             (
               ((By - Sy)*(R2x-Sx) >= (R2y+Rr-Sy)*(Bx-Sx) & R2x - Sx >= 0 |
                (By - Sy)*(R2x-Sx) <= (R2y+Rr-Sy)*(Bx-Sx) & R2x - Sx < 0) &
               ((By - Sy)*(R2x-Sx) <= (R2y-Rr-Sy)*(Bx-Sx) & R2x - Sx >= 0 |
                (By - Sy)*(R2x-Sx) >= (R2y-Rr-Sy)*(Bx-Sx) & R2x - Sx < 0)
             )
            )
          )
        )
       )

       }@invariant(
         Bdx = Bdx0 & Bdy = Bdy0 & Bs = Bs0,
         /* on the line connecting the source to the destination */
         ((By-Sy)*(Dx-Sx) = (Dy-Sy)*(Bx-Sx))
       )
     ++
     /* Ball evolution: case (B): ball stationary */
     {
      Bx' = Bs*Bdx, By' = Bs*Bdy, t' = 1,
      (Bs*Bdx)^2 + (Bs*Bdy)^2 = 0
     }@invariant(Bdx = Bdx0 & Bdy = Bdy0 & Bs = Bs0 & Bx = Bx0 & By = By0)
    )
)*@invariant(
    /* In order to be able to hide this before applying global rule invariant
       rule, need to duplicate these initial conditions here explicitly */
    (R1x - Rr > R2x + Rr | R2x - Rr > R1x + Rr) &

    /* Strong statement: we can never be in the reflection situation */
    ( Bdx = 0 & Bdy = 0 | /* doesn't apply if it's stationary */
      (
       ((Bx != R1x | (By < R1y - Rr | By > R1y + Rr)) | (Bx = R1x & By = R1y)) &
      ((Bx != R2x | (By < R2y - Rr | By > R2y + Rr)) | (Bx = R2x & By = R2y))
     )
    ) &

    /* Strong statement: always en-route between robots */
    (
     (Sx = R1x & Sy = R1y |
      Sx = R2x & Sy = R2y) &
     (Dx = R1x & Dy = R1y |
      Dx = R2x & Dy = R2y)
    ) &

    /* The kicker should never choose itself as the destination */
    (Sx != Dx | Sy != Dy) &

    Bs >= 0 & Bs <= Bsk &
    (
      Bdx = 0 & Bdy = 0 |
      (
       /* When moving ... */
       /* ... is somewhere on the line from src to dest */
       (By-Sy)*(Dx-Sx) = (Dy-Sy)*(Bx-Sx) &
       /* ... inside the segment between src and dest
               NOTE: no vertical movement allowed) */
       (Bdx > 0 & Bx >= Sx | Bdx < 0 & Bx <= Sx) &
       (Bdx > 0 & Bx <= Dx | Bdx < 0 & Bx >= Dx) &
       /* ... is moving along the src-dest line (cross product of velocity
              vector and src-dest vector should be 0, i.e. parallel) */
       (Dx - Sx) * Bdy - (Dy - Sy) * Bdx = 0
      )
    ) &
    (
     /* Ball within field boundaries... */
     -F < Bx & Bx < F & -F < By & By < F
    )
  )
\](
    /* Ball stays within field boundaries... */
    -F < Bx & Bx < F & -F < By & By < F
  )
}
```