



RoboFesta-MK

RoboCup Junior Football

Programming Checklist



Although these notes will not give you a complete solution to the problem of programming your robot footballer, they do provide a set of programming checkpoints that might prove useful to you as you develop your own programs. The order in which the checkpoints are presented is not necessarily indicative of the order in which you should try to attain them. You may find that several people can develop different parts of the program independently of each other at first.

The notes assume a robot with two motors, one driving each wheel; one forward facing bumper connected to a touch switch, and two light sensors - one forward looking and one downward looking.

These notes are intended to be independent of any particular programming language. Language specific hints and tips that illustrate some of the programming constructs described here are provided in two separate documents, one concentrating on Robolab, one on the Mindstorms (RCX language) programming environment.

The checkpoints are based on the Robolab sample football player program developed by Brain Thomas and Ian Maud for RoboCup Junior Australia. This program, along with plans for building a suitable robot, are available from the RoboCup Junior Australia website:

<http://www.robocupjunior.org.au/>

Register yourself and use the password they send you to enter the Buildabot area.

Before you start to construct your programs, read through the whole of this document. You will find some useful hints at the end about creating small program fragments (such as subroutines) that you can reuse in larger programs.

When writing your programs, remember that in each match your robot will play one half of the game attacking the goal at the dark end of the pitch, and one half of the game attacking the goal at the light end of the pitch. If your robot makes light readings from the grayscale pitch to work out which direction it is going in, or alter its tactics, you will need to find a way of coping with this. The simplest way would be to create two programs, one for playing each way. However, you may be able to come up with a more elegant solution that just requires you to set the direction the robot is playing at the start of the program, and use this information to modify the robot's behaviour when necessary.

You should also bear in mind that even if your robot is not taking a kick-off at the start of a half, it may have to take a free kick during the half. Again, you might have a couple of programs on your robot in different program slots, one that starts with a kick off, the other that receives the kick off.

For specific hints and tips on programming in Robolab or Mindstorms/RCX language, please refer to language specific documents provided.

Checkpoint 1 - the blind kick-off:

Quoting from the rules (<http://satchmo.cs.columbia.edu/rcj/rcj2002/soccer.html>):

5.4. Kick-Offs.

5.4.1. Each half of the game begins with a kick-off.

5.4.2. All robots must be in located on their own side of the field.

5.4.3. All robots must be halted.

5.4.4. The ball is positioned by the referee in the center of the field.

5.4.5. All robots on the team not kicking off must be at least 15cm away from the ball.

5.4.6. The team not kicking off places their robots on the field first.

5.4.7. The team kicking off will place one robot at least 5cm away from the ball.

5.4.8. The referee may adjust the placement of the robots.

5.4.9. On the referee's whistle, the robot kicking off will be started by a human team member.

Hint: This robot should then strike the ball.

5.4.10. The robot kicking-off cannot re-contact the ball until 1 second has elapsed.

5.4.11. After the robot kicking-off has contacted the ball (or 5 seconds has passed without that robot contacting the ball), the referee will blow the whistle again and the other robots will be started by remote control or human team members.

5.4.12. Robots that start before the whistle will be penalized and removed from game play by the referee for 1 minute.

Place the robot that will take the kick-off directly facing the ball. In order to make the kick off:

- a) command the robot to drive forward for x seconds, where x is long enough for the robot to travel the 5cm or so to the ball (rule 5.4.7);
- b) assuming the ball is going away from the robot taking the kick off, either pause the robot for 1 second or make it reverse (rule 5.4.10).

Note that in this strategy, the robot does not sense where the football is - it simply drives forward for a short period of time in the hope that it will blindly make contact with the ball.

If you are sure that your robot will hit the ball well within the required 5 seconds of starting the robot, you may want to put in a short delay *before* the robot moves. This will give yourself time to move your hand away from the robot after pressing the run button.

Checkpoint 2 - making for the ball following the kick-off

The team receiving the kick off must place their robot at least 15 cm away from the starting position of the ball (rule 5.4.5). This robot is started once the ball has been kicked off (rule 5.4.11) and can immediately make for the ball. (The robot who took the kick-off can also make for the ball after about 1 second following the kick-off). This will require the robot to use its light sensor to identify where the ball is. Because you have only one light sensor, your robot will have to perform some sort of scanning behaviour. That is, it will look from left to right for example, looking for a high reading from the light sensor. This will require several mini-checkpoints to be attained:

Checkpoint 2.1 - doing a single twirl

The scanning behaviour can be achieved by getting the robot to turn, ideally on the spot (how you constructed your robot will determine how well it can do this).

Program your robot so that it makes a complete turn. One way of getting robots to turn is to drive one motor forward and one motor backwards. So for example you might turn the right motor on forward, and the left motor on backwards, for 1.5 seconds to get a complete turn.

Checkpoint 2.2 - doing a twirl in two steps

Now program your robot so that it makes half a turn, pauses, and then makes another half a turn. You could use this action as part of a behaviour in which the robot looks to see if the ball is ahead, or if not turns to see if the ball is behind it.

Checkpoint 2.3 - doing a twirl in many steps

See if you can make your robot do a complete rotation using 4 or more partial turns, each followed by a brief pause. This sort of stepped scanning action can be used as part of behaviour in which the robot more closely identifies the whereabouts of the ball. The maximum number of partial turns in a single revolution will be determined by the speed at which your robot turns - the shortest period of time the motors can be turned on for is 0.1s.

Checkpoint 2.4 - seeing the ball

Okay, so now you can get your robot to do a twirl. What you now need it to do is see the ball. If you are scanning for the ball, how will you locate it? Suppose it is to the right of you. First, without moving, take a measurement from a forward looking light sensor; suppose it reads the value x . Now scan right (i.e. turn on the spot right a little way), pause, and take another reading from the light sensor: say it reads the value y . If the robot is now looking towards the ball, the value from the light sensor will have increased (that is y will be greater than x : $y > x$). Note that the RCX brick will only take a reading once every 0.1s, so you may have to insert a small delay before each reading. You may want to check the value from the light sensor against a relatively high threshold value to make sure that you are actually seeing the ball, rather than just detecting ambient light. If the reading does pass the level of the 'background' light, you could now move towards the ball.

Checkpoint 2.5 - seeing the ball, better?

You could refine the behaviour described in checkpoint 2.4 by continuing to turn until the value from the forward looking light sensor drops. Suppose I'm looking straight at the ball, which isn't moving. Now, if I make a turn, the reading from the sensor will go down, which means I have to turn back to face the ball. Why would it make sense to keep on turning as the value from the sensor increases, until it starts to decrease again?

Checkpoint 2.6 - making for the ball

So you've scanned until you think you know where the ball is - now all you have to do is drive forwards towards it, don't you (or did your robot turn past the ball as part of the scanning procedure)?

Checkpoint 2.7 - didn't see the ball?

If your robot made a complete turn while scanning for the ball, and didn't see it, then it may be too far away. Get your robot to move somewhere else on the pitch, and then start scanning again. If you know how many partial turns it takes the robot to complete a full turn, and you somehow keep count of the number of partial turns your robot has made during a scan, you should be able to work out when your robot has done a complete turn.

Checkpoint 3 - where's your robot on the pitch?

As well as the forward looking light sensor, there is a downward looking one that your robot can use, in conjunction with the gray-scale pitch, to identify which way the goal is. You can use a scanning behaviour to detect this too.

Checkpoint 3.1 - am I going the right way?

The simplest approach to working out whether you are going the right way is to take a sample reading from the downward looking light sensor, store this value (value x , say) in a container, move forward, and take another reading (value y , say). If you want to go towards the dark end, you know you're on the right lines if x is greater than y . That is, the first reading detected a brighter part of the pitch. Similarly, if you want to go to the light end of the pitch,

you would want $x < y$. If you're going in the wrong direction, either reverse or do a half turn and then go forward (you'll be facing the other way now).

Checkpoint 3.2 - shoegazing twirl

Depending on the light conditions, you may be able to detect changes from the downward looking light sensor while doing a twirl or a turn. As with the scan for the ball, you can keep turning as the reading increases, say, (if you want to go to the light end of the pitch), stop as the reading decreases again, turn back a bit then head off straight towards the light end.

Checkpoint 4 - avoid the walls/other robots

One thing you don't want your robot to do is to keep on trying to move forwards if it has run up against the wall or another robot.

If you detect the bumper being pressed, get your robot to back up a little way, make a turn (left or right), then move forwards again.

However, when building the bumper, bear in mind that you don't want the robot to think the ball is the wall, and run away each time it comes into contact with it

Checkpoint 5 - calibrate the robot

It can be useful to calibrate your robot so that it has some chance of interpreting its sensor readings sensibly. In particular, you will probably have to calibrate the light sensor thresholds for each pitch your robot will be playing on to take into account the particular background light conditions.

The simplest way to calibrate the settings is by trial and error. Set the threshold to an arbitrary value, test your robot, then modify the thresholds if it doesn't appear to behave properly.

A more sophisticated approach is to calculate the threshold based on some limiting/extreme case values as measured by the light sensor. For example, possible measurements you might make are:

- the value of the downward looking light sensor at the dark end of the pitch;
- the value of the downward looking light sensor at the light end of the pitch;
- the value of the downward looking light sensor at the centre of the pitch;
- the value of the forward looking light sensor with the ball absent;
- the value of the forward looking light sensor with the ball next to it;
- the value of the forward looking light sensor with the ball 5cm or so away from it.

You can manually identify these values by downloading a simple program to your robot that sets up the two light sensors on the appropriate RCX input ports. Run the program, and then use the View button on the RCX to move the little arrow that appears next to the first input port to the input port your light sensor is connected to. The reading on the screen is the value being returned from the light sensor. Place the robot at the various sampling points identified above, and record the readings. You can then use these values to help you calculate the various threshold values used by your program.

If you are really ambitious, you might store the values in specific containers before the robot starts to play football and then calculate the thresholds automatically. To store the values, you will have to have a calibration routine at the start of your program. You can use Wait for commands to help you do this. For example: place your robot at the dark end of the pitch. Start your program with a Wait for Touch sensor command. When you press the touch sensor, take the reading from the downward light sensor and store it in the container you are

using for 'dark end of pitch' then make a beeping sound to say your robot is ready for the next part of the calibration process. Move your robot to the light end of the pitch. Use another Wait for Touch sensor command in your program, followed by a sequence that takes the reading from the downward light sensor and stores it in the container you are using for 'light end of pitch'. By producing a checklist of calibration steps that follows your program, you can calibrate the robot with useful readings. Once the robot is calibrated, have it play a tune and it should be ready to play football. Pressing the bumper again should start the robot actually playing the game. You will also have to work out some way of pausing the robot so that it can be restarted to either take or defend against a free kick.

Checkpoint 6 - tactical play

If you are trying to score in the goal at the light end of the pitch, and your robot has seen the ball but it is towards the dark end of the pitch, relative to the robot, then your robot shouldn't drive straight towards the ball in case it scores an own goal.

Checkpoint 6.1 - only search forwards...

...where forwards means towards the other team's goal. If you calibrate your robot before the start of the game, your robot should know when it's at the end of the pitch. If it gets to the end of the pitch without locating the ball, it can turn round, go back to it's own end, and start searching for the ball in a forward looking direction again.

Checkpoint 6.2 - swerve...

If your robot has seen the ball ahead of it and is moving towards it, check the downward looking sensors to make sure you're going in the right direction. If you are, all well and good. If you are heading towards your own goal, make the robot do three sides of a square to get behind the ball: do a right (or left) hand turn, go forward a bit, do a left (or right) hand turn to straighten up, go forward again, do a left (or right) turn. Now do a left (or right) hand scan, find the ball and rush forwards to score in a blaze of glory.

Useful program blocks

You may find it useful to use subroutines of one sort or another that can be reused throughout your program and help to keep it looking tidy.

The following are likely to be particularly useful:

<i>Go forwards</i>	- turn both motors on in the forwards direction
<i>Go backwards</i>	- turn both motors on in the reverse direction
<i>Stop</i>	- turn off both motors
<i>Go forwards for ...</i>	- turn both motors on in the forwards direction for a specified time and then stop them
<i>Go backwards for ...</i>	- turn both motors on in the reverse direction for a specified time and then stop them
<i>Turn left</i>	- turn the left motor on forwards and the right motor on reverse
<i>Turn right</i>	- turn the right motor on forwards and the left motor on reverse
<i>Turn 180 degrees</i>	- turn one motor on forwards and the other on reverse for a predetermined time (identified through testing), then stop. You could refine this by having both clockwise and anti-clockwise turns defined.
<i>Turn 90 degrees left</i>	- turn left for a predetermined time, then stop
<i>Turn 90 degrees right</i>	- turn right for a predetermined time, then stop

<i>Turn x degrees right/left</i>	- turn right/left for a user-defined time, then stop
<i>Left-hand swerve</i>	- turn 90 degrees left, go forward for an appropriate time, turn 90 degrees right, go forward, turn 90 degrees right, then stop
<i>Right-hand swerve</i>	- turn 90 degrees right, go forward for an appropriate time, turn 90 degrees left, go forward, turn 90 degrees left, then stop
<i>Ball detected?</i>	- is the forward light sensor is above a the threshold for positively identifying the ball?
<i>Scan for ball left</i>	- if the ball isn't detected, make a partial left turn, and try to detect the ball again; if the ball isn't detected, make another partial turn left.
<i>Scan for ball right</i>	- if the ball isn't detected, make a partial right turn, and try to detect the ball again; if the ball isn't detected, make another partial turn right.
<i>Go to ball</i>	- if the ball is detected ahead, go straight until the ball ceases to be detected
<i>Look for ball</i>	- scan for the ball; if a complete rotation doesn't locate the ball, go somewhere else on the pitch and scan again.
<i>Go to lighter end of pitch</i>	- sample the downward looking sensor, move forward a short way, and sample again. If the first reading is significantly less than the second, you are going the right way. Otherwise, turn round 180 degrees and try again. If that doesn't work (for example, the robot is facing across the pitch), turn 90 degrees and try again. And if that doesn't work, turn another 180 degrees, and hopefully the robot will now be facing the correct direction. Alternatively, you might try to spin the robot and have it move forward when the downward light sensor value stops increasing.
<i>Go to darker end of pitch</i>	- similar to the previous, but now you want lower values from the light sensor as you go forward.

Putting it all together

Constructing the overall program is likely to present you with a major challenge. The overall structure of your program will be along the lines of:

```

Start
    [Take kick-off]
    Play match
Stop

```

One of the most straightforward ways of approaching 'Play match' is to produce a linear program that executes each behaviour in turn, and which is itself contained in a loop. For example:

```

Play match:
    Begin loop
        Scan for ball
        Check direction of goal
        Go forward to ball OR [swerve round ball (perhaps scan for ball?)]
        While you can see the ball, keep going forward
    Go back to start of loop

```

We would not expect you to get much further than this in the short time you have had available for the competition. However, it is possible to use more complicated programs that use several tasks that all run at the same time (that is, in parallel, or concurrently, with each other). Within each task, you might have a loop that tests one of the sensors and takes an appropriate action. For example, you might have a task 'avoid obstacle' that loops through 'if bumper pressed, reverse and turn'.

One problem with using several tasks that execute in parallel is that different tasks may be trying to tell the motors to do different things at the same time. One way round this is to prioritise tasks such that the task with the highest priority stops all the other tasks when its conditions are met. For example, an avoidance behaviour might tell the robot to reverse when the front bumper/touch sensor is pressed. However, before it takes this action, it might be appropriate to stop the 'go forward' task which makes sure that both the motors are driving forwards. Once the avoidance behaviour has completed, then it can restart the go forward task. The avoidance behaviour is thus seen to be effectively acting at a higher priority than the go forward task.

Having to stop and start different tasks from within a task that runs at high priority can become very complicated very quickly. One way round this is for each task to *request* a particular set of motor outputs and have another *arbitrator* task choose which set of motor outputs will actually be used. The arbitrator knows the priority of each of the other tasks, so when several tasks request different motor actions, the arbitrator sets the motors running in the way requested by the task with the highest priority. This approach is known as a subsumption architecture and is the basis for many research grade robots today. If you do want more information about this approach, you can find more details at: <http://www.restena.lu/convict/Jeunes/Subsumption.htm> - the page provides a brief overview of the approach and also describes a Robolab program that implements a subsumption architecture arbitrator that you can download.

robofesta@open.ac.uk