CSC C85 – Fundamentals of Robotics and Automated Systems
Final Project – Lego-EV3 Robo Soccer

This is it! Time to take everything you have learned and put it into use to create the mightiest LEGO soccer-playing bot the world has ever seen! This project will require you to work with sensors, actuators, remote communications, planning and AI, localization, and a dynamic environment that changes in real time.

Learning Objectives – after completing this project you should be able to:

Integrate your expertise with sensor management, control, robust real-time software and simple A.I. to implement the software that runs a little soccer-playing bot

Write software to handle noisy dynamic inputs under uncertainty, and in the presence of an opponent

Use prediction as a component of your robot's planning process. Anticipate the actions of your opponent and the movement of the ball, and have your bot act accordingly.

Use visual information – vision is a fundamental source of data about the world. You will learn to use a very simple vision-based system while dealing with its uncertainties and limitations.

Design, implement, and test a simple state-based A.I. to drive your robot's behaviour.

And in case you didn't already know: You will discover that soccer is a beautiful sport!

Skills Developed:

Working with a multi-component system. Sensors, processing, and actuators are not on the same platform.

Writing code that performs carefully planned actions in the presence of inaccurate information and noise.

Developing state-based AIs. Writing code that reacts appropriately to a changing environment.

Working with visual inputs, along with their limitations and uncertainties.

Reference material:

Your lecture notes on control systems, sensors, and reliable software design

The comments and explanations in the starter code.

As always: *Be ingenious and creative. Bonus points for clever solutions to tough problems.*

Important Note: This project is designed for Linux. The starter code ***will not compile or run on Windows/Mac***. It also requires a playing surface and an EV3 bot for most of the testing and development. Therefore, you're strongly advised to work at the IC308 Lab either in one of the workstations or on your own Linux laptop.

Why Robo Soccer?

There is a number of reasons why soccer is a very well suited task for testing embedded system ideas.

- It is a real-world task that involves complex actions carried out with precision and speed.

- It imposes real time constraints on the embedded system which must respond appropriately to changes in the configuration of the play field. This forces the system to process incoming data and decide on suitable actions within a short time interval.

- It requires the use of more advanced sensors; these sensors provide richer information than we have used thus far, but are nevertheless noisy and their data requires careful analysis and interpretation.

- It requires integration of all the concepts we have studied thus far: code optimization, sensor and noise management, localization, building reliable software, responsing in real-time to sensor input, and AI/planning.

- Soccer-playing robots are not something you see every day!

This project provides you with an opportunity to bring together all the tools you have acquired during the term, strengthen and improve your understanding of course concepts, and apply the skills you have been developing over the past weeks toward solving a challenging problem.

Make the most of this chance! Not many projects during your academic career will be as challenging or demanding, but also not many will be as satisfying! Go make us proud!

Acknowledgements:

The project starter code and Robot Control API were updated to work with the EV3 Lego robots in the Summer of 2018

Robot Control API and updated build chain – by Lioudmila Tishkina
Image processing core, robo-soccer and AI starter by – Francisco Estrada
Robo-soccer A.I. starter – Per Parker

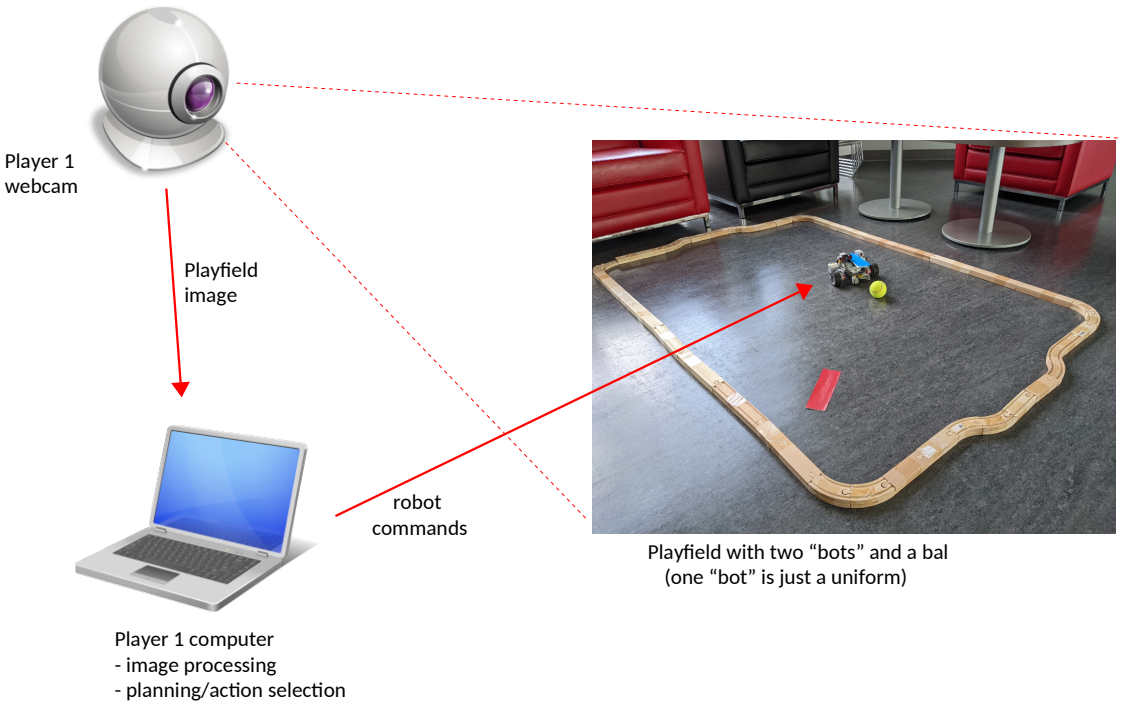
Project Overview

Just like with your robot localization project, we will be using the EV3 kits as remote bots controlled via bluetooth from your laptop.

For soccer playing, the robot must be able to perceive the playing field and determine the locations of itself, the opponent, and the ball in order to determine its course of action. To achieve this, we will use a webcam, looking at the playifield from an elevated position. The camera will provide visual input which is converted in to a set of input data your robot will use to play soccer.

Remember that you can poll the EV3 sensors remotely, there is no reason why you can't or shouldn't use the sensors on the bot to make your system better.

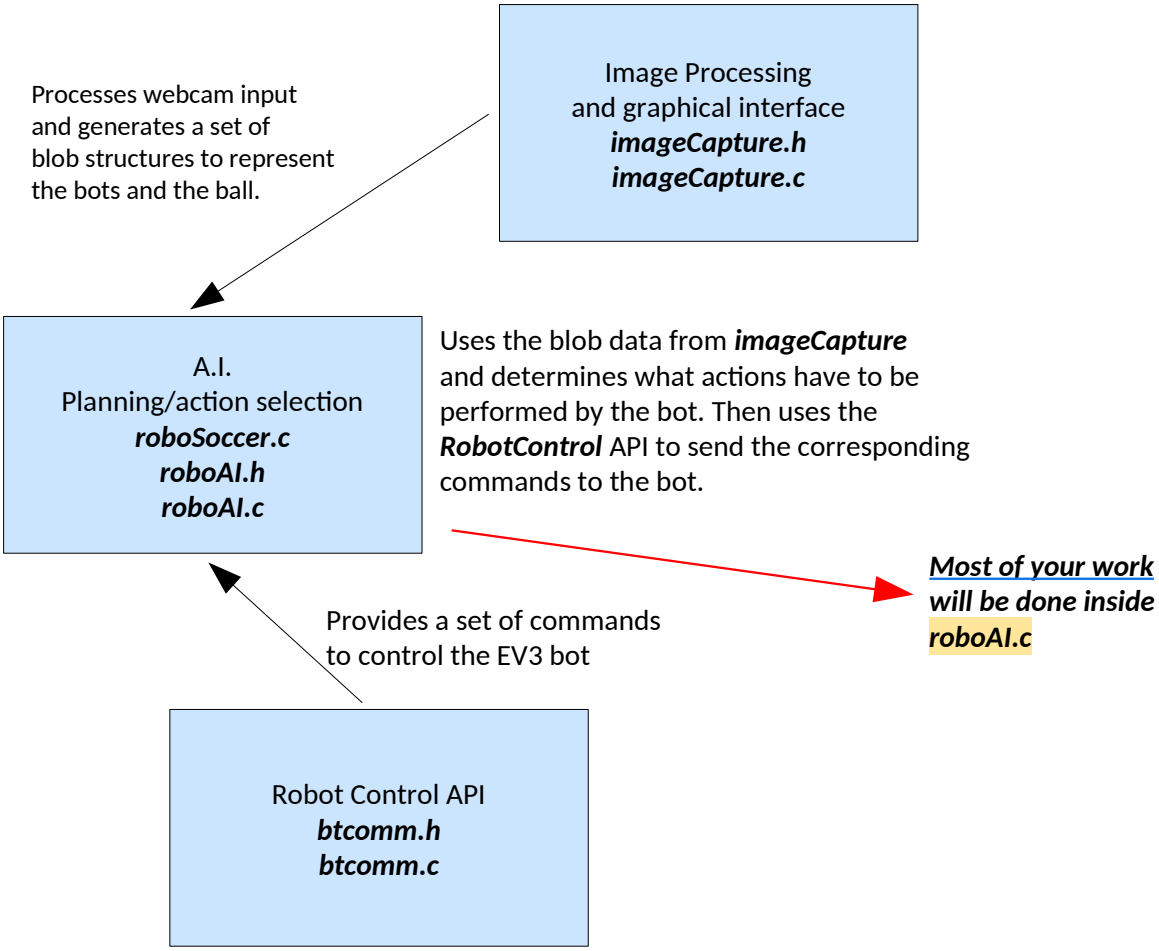
- The EV3 bots will wear coloured uniforms, and we use a yellow ball. This makes it relatively easy to visually identify and locate the agents in the field.
- Your will be in charge of the robot AI code that will determine at each moment what commands to send to the bot so that it can play soccer. The overall system is shown below.



Games are 1-on-1, each player uses their own laptop and webcam.

Structure of the Software Distribution

Download and uncompress the starter code. It will generate a ***RoboSoccer_EV3*** directory with all the modules that comprise the project code. The general structure of the code is as shown below (files are within *RoboSoccer_EV3/src/*):



What you need to read and understand:

imageCapture.h: You must understand what the ***blob*** data structure contains so you can use it.
btcomm.[h,c]: These are the controls you have at your disposal for controlling the bot.
RoboSoccer.c: This is the program entry point. ***You will need to change the EV3 hex ID at the top of this file to match your bot's bluetooth ID.***

Compile the code with the 'robo_soccer_compile.sh' script, and run

./roboSoccer /dev/video1 0 0 (you may need to change the video device)

Setting things up

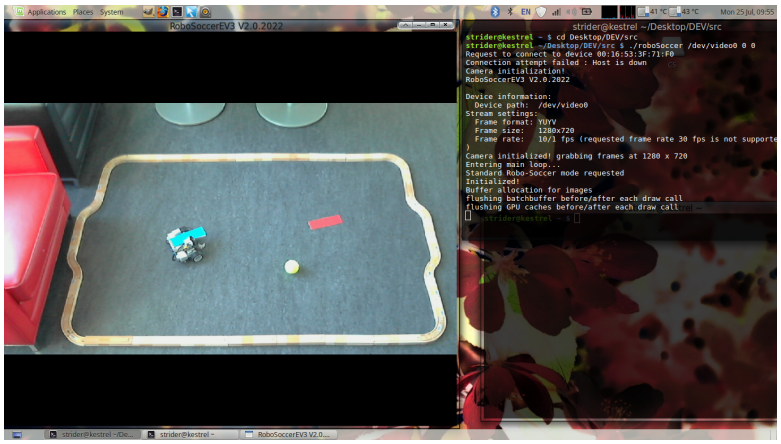
1) Set up your play field

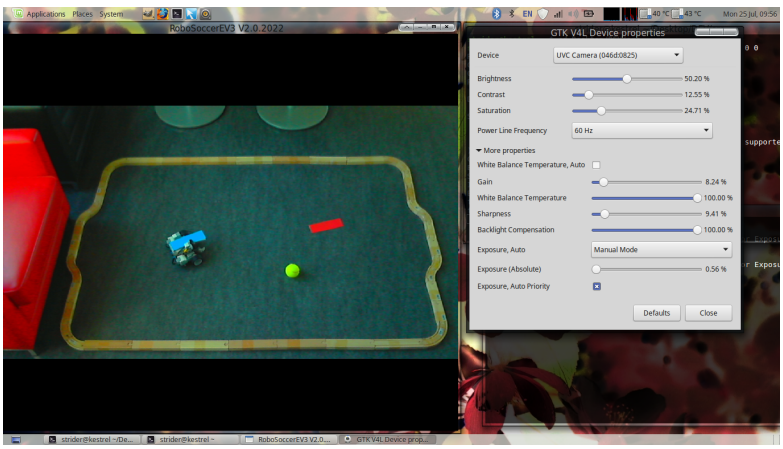
You will need a rectangular playing field. Set it up on a neutral coloured surface. Ideally use a gray surface similar to the colour of the classroom floors and lab at IC. This is where your bot will play during competition day. ***The field we'll be using is a rectangle 170 cm x 115 in in size.***

You can use non-marking tape or any other form of delimiter on the lab floor, but try to make it rectangular and large enough that your bot will have space to run/turn and the ball can bounce around a bit.

2) Set up the webcam

Place the webcam somewhere high overlooking the entire field. Tune the camera's exposure, saturation, and brightness so you can clearly see the uniforms and ball. use '***gtk-v4l***' or '***v4l2ucp***' (depending on your Linux distro) for this purpose.





The top image on the left shows the view immediately after calling the robo-soccer code, the camera's auto-adjust has made everything too bright and colours are washed out. Some regions in the image are over-exposed.

The bottom image shows the view after adjusting parameters. Note that ***you will see different controls*** depending on your webcam and software, and the values required to get a clean image will be different too.

You want to clearly see the uniforms and ball, with no glare. take some time to become familiar with this process since everything else depends on having a clean image to work with.

Setting things up … continued

3) Colour calibration

Because we are running this in un-controlled conditions, we have to calibrate the camera input to account for changes in illumination, the colour of the floor, furniture and other items around the field and even people walking around while we're running the software.

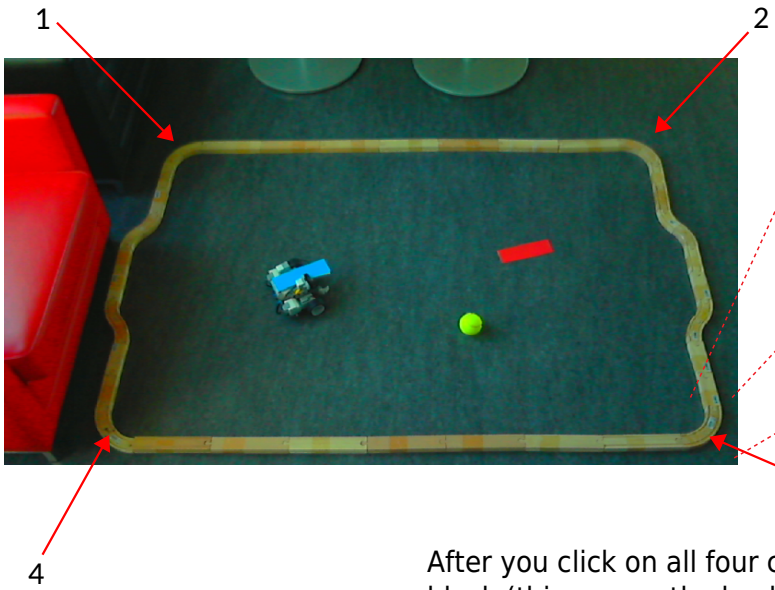
The first step is to tell the image capture code what the colours of the different agents look like. Press ***'c'*** on the GUI to bring up a cross-hair which you can control with the keys 'a', 's', 'd', 'w' (if you use the shift key along with these, the crosshair moves by 5 pixels instead of 1).

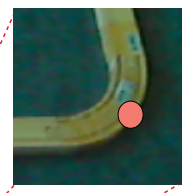
Place the crosshair on the ***blue robot and press space*** to capture a colour sample
Place the crosshair on the ***red robot and press space*** to capture a colour sample
Place the corsshair on the ***yellow ball and press space*** to capture a colour sample
Finally, place the crosshair somewhere on the background and press space.

4) Mark the corners of the playfield, and get a background image

Now we have to tell image capture where the corners of the field are (since this depends on where the camera and field were placed, and will change for sure.

Remove the bots and ball from the field, we want to capture the background appearance. Then press ***'m'*** in the GUI to bring up the crosshair and use the same Process as above to capture the location of the ***top-left, top-right, bottom-right, and bottom-left*** corners of the field, in that order.





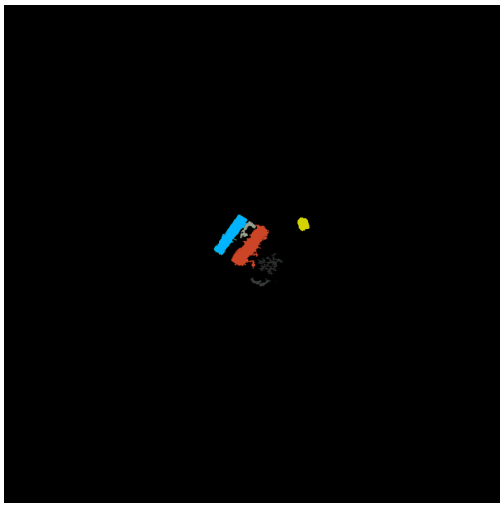
Try to be as accurate as you can when capturing the corner locations.

After you click on all four corners, wait until the image goes black (this means the background has been captured)

5) Fine-tuning the blob detection process

At this point the playfield initialization is complete.

There's two more calibration steps to complete, ***bring the robots and ball back into the field.*** You should see an image like the one shown below:



You may see more stuff, or not see some of the blobs or blobs may be broken up.

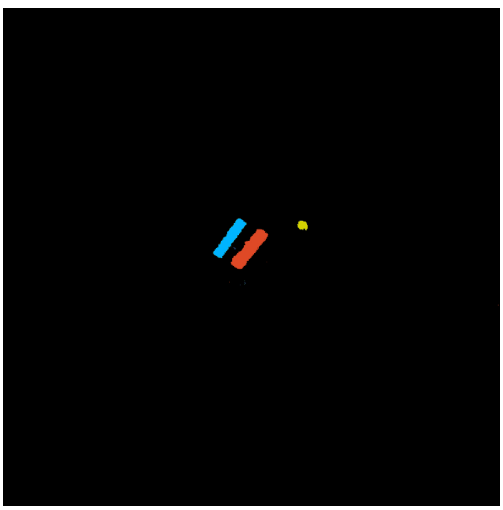
The GUI provides controls to adjust the image capture to get clean blobs:

'<' and '>' to change background subtraction threshold. Use this if the image is missing things or showing too much junk.

'[' and ']' to change the colour satutarion threshold. Use this if the image has a lot of non-uniform or ball regions, or if it's missing those regions.

'{' and '} to adjust the colour angle threshold used to group pixels into colour blobs. Use this if the blobs have too many holes or if they are adding non-blob regions.

Play with these adjustments until you understand how they affect the resulting image, then adjust them until you have ***mostly clean blobs for the bots and ball.*** it is impossible to get perfect blobs! So don't try – but you should see the two bots and the ball clearly as in the image below.



These adjustments are always available, so you can fine-tune them while the code is running.

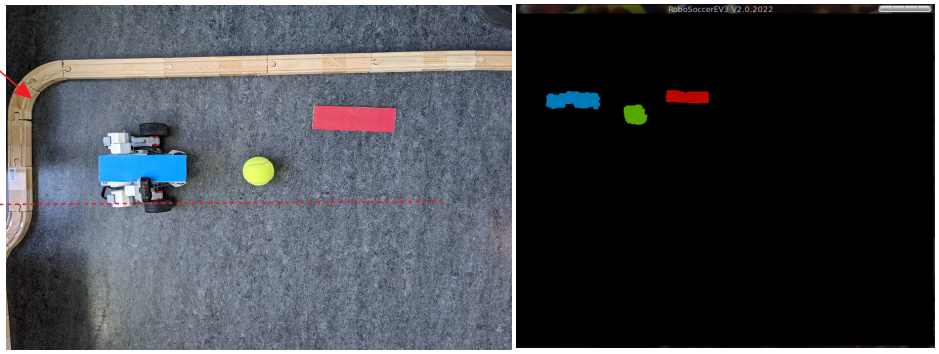
We have one final calibration step, and then we're ready to play!

6) ***Perspective projection adjustment* calibration.**

The final calibration step is incredibly important. To see why, place the two bots (it's fine if you use a simple uniform and your bot when you're testing on your own) and the ball and line them up somewhere close to the ***top of the field*** as shown in the image below.

Top-left corner of playfield

Robot lined up with the ball (I don't have Two, so I placed The red one at A spot that makes its uniform line-up with the blue one on the blob image)



Notice that even though the blue bot and the ball are aligned on the field, they don't appear aligned on the processed blob image. This is because of perspective projection, because the uniform is not directly on the floor, from the camera's viewpoint, it appears as if the blue bot is further up in the image than it should be. ***We have to correct this*** or else your program will never be able to tell if the bot is aligned with the ball.

Press 'z' on the GUI – this will show tracking boxes for the bots and ball. Then press ***'x'*** to capture the relative locations of the bots and the ball for this part of the field. Now align the bots and the ball in the same fashion, but ***near the bottom of the field***. Then press ***'x'*** a final time to capture relative positions near the bottom.

After calibration, the blobs should line up nicely (the middle image above shows them before, the image on the right shows after calibration). If they don't line up properly, re-do the calibration. This step has to be done carefully!

You only need to do calibration once. Afterwards, just press ***'g' to load all calibration values when you start the program!***

Understanding the Information Flow

The blob data structure – Here is all the sensing data you need

Your code in ***roboAI.c*** does not have direct access to the image processing data. This is intentional. ***You do not have to look into the image processing code to complete your project.*** The only information your A.I. needs is what is provided by the image processing code in the form of a ***linked list of blob data structures.***

This linked list, imaginatively called '***blobs***' in ***AI_main(),*** contains a set of blobs representing recent objects found on the playing field.

Things to note: ***There may be a large number of blobs, certainly more than 3*** Blobs corresponding to the bots/ball must be identified among those in the list (more on this later).

While you are allowed to change the internal values of the blobs in the list, ***you are not allowed to modify the list itself by removing or inserting blobs.*** This list is controlled by imageCapture.c
For each blob, the following data of relevance to you is stored:

cx, cy, holds the current location of the blob's center.
vx, vy, holds the current velocity vector for this blob. This is updated by the AI code. ***the velocity vector is not valid if the bot is rotating in-place.***
mx, my, heading direction as a unit vector. Last known heading for this blob. This vector remains constant if the blob is not moving. ***This may not be valid during rotation.***
dx, dy, direction vector for the blob, points along the long side of the rectangular uniform of your bot. ***This is valid at all times, but can point backward.***
size, x1, y1, x2, y2, size of the blob (in pixels) and coordinates of the bounding box as top-left and bottom-right.
R,G,B, average colour of pixels on the blob.
idtype, identifier for this blob. Set by the A.I. code that tracks agents. See the relevant part of the starter for details.

Other data in the blob structure are used by the image processing code, However, the data listed above should be all that is needed to solve this project.

Be sure to look into imageCapture.h and familiarize yourself with the blob data structure. Then look into roboAI.h and roboAI.c and see how blobs are being used in the code provided.

The data inside the bot/ball blobs, and the data in the AI data structure is all you need to plan your strategy. Make sure you use all of it smartly.

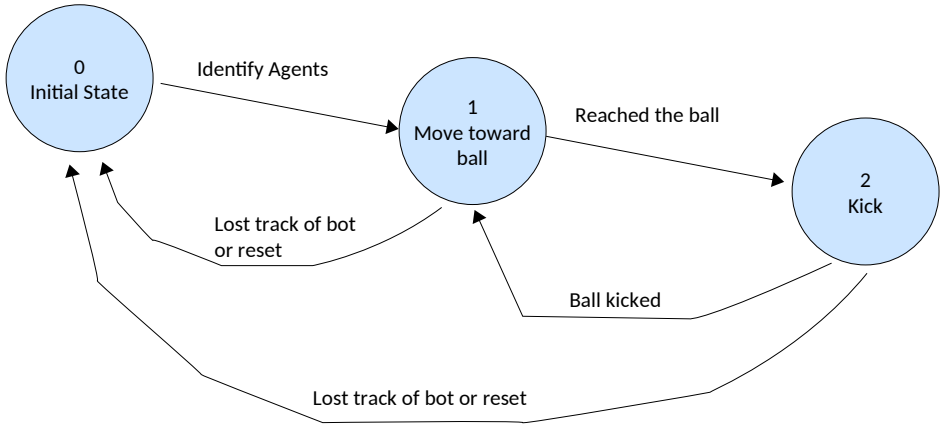
Understanding the Information Flow

The AI module – Here's where you do your work

You will be working most of the time within *roboAI.c* and ***roboAI.h*** to implement the data processing, planning, and game playing logic.

Your bot will use a simple ***state-based A.I.*** which is nothing more fancy than a finite state machine which has a number of possible states for the robot to be in. Each state is associated with specific actions to be carried out, and events in the field trigger state transitions.

As a simple example, consider the following mini A.I. for a very simple player:



This player is constantly chasing the ball, and once it reaches the ball, the ball gets kicked. This simple A.I. does not consider a second player and its actions, or where the ball is w.r.t. the goal; so your own A.I. will be significantly more complex, but the idea is the same.

Each state is associated with one or more functions within the code. ***AI_main()*** will be responsible for calling the appropriate function given the current state of the A.I.

The bot starts at states 0, 100, or 200, and the starter code provides the Functions needed to perform agent ID and take the A.I. to the next state (1 in the diagram above, 1 or 101 or 201 in the code depending on the game mode).

You are not allowed to change the code that handles the transition from state zero to the next state in the A.I.

Understanding the Information Flow

Data stored by the A.I.

Within ***roboAI.h*** you will find the ***AI_data*** structure which contains all the information kept by the A.I. and used to play the game.

You must read the descriptions in ***roboAI.h***, but for now, here's a summary of what is contained there:

- A flag indicating which of the sides of the field belongs to the bot

- A flag indicating the colour of the bot's uniform

- The current A.I. State

- Motion flags that can be used to keep track of what the robot is doing

- Flags to indicate whether the bot, its opponent, and the ball have been identified

- Pointers to the blobs for the bot, the opponent, and the ball so you can access the data stored within the corresponding blob data structures.

- ***Position of the bot/opponent/ball on the previous frame***

- ***Current velocity for bot/opponent/ball***

- ***Current heading vector for bot/opponent/ball***

You will have to use the information within the ***AI_data*** structure to determine what state transitions must be carried out, and what actions must take place.

You can add data to the AI_data structure, but make sure there is a good reason for doing so. We will penalize superfluous additions that only bloat the A.I.

Keep in mind: Sometimes blobs disappear from one frame to the next, for example, The ball may go behind a bot and become invisible, or they can disappear simply because of a bad frame from the camera. Your code should be able to handle this Without breaking or producing a segfault.

Understanding the Information Flow

In order to continue at this point, you need to have a working bot.

You are free to design your robot as you wish. But you may want to consider the following:

* If your bot is too big, you will have problems moving around the opponent and getting to the ball.

* Your bot should be able to wear either the blue or red uniform. And you have to be able to change the colour quickly since the colours will be assigned randomly for each actual match. Be sure to test all your work with both colours!

* Don't make the bot too tall! Remember that there is a localization error due to the offset between the bot's uniform and the floor.

* Make sure as much of the uniform's colour is visible as possible.

* Connect the motors to the correct ports! See the documentation for the API

* The uniform's center should be located on the ***center of rotation of the bot***. Otherwise, when the bot rotates, it will look like it is also moving!

Once you have a working bot, you must pair it up with the computer running your code via Bluetooth.

1) The easiest way to do this is to turn on your bot, and run the ***robo_soccer*** program, then follow the instructions on your bot (and laptop) to pair the two. This need only be done once, afterwards your laptop will remember the bot you are using.

   If this doesn't work, try using the ***blueman applet*** which is part of one of the many Linux libraries for handling bluetooth – your Linux distribution may have a different tool, but we've found blueman applet to work most of the time.

2) Note the EV3's HEX key identifier. This ***must be placed at the top of RoboSoccer.c*** so that the code can talk to your bot.

   Re-compile and run the code. The starter code will print out a message if it is not able to connect to your bot. Please check the bot is paired with the computer, the correct hex code is used in the ***roboSoccer.c,*** and your bot is ***powered on***.

   If no error occurs, your bot is now under the control of ***roboSoccer.c***

Starting up the A.I.

Having successfully completed the initialization steps in the previous page, give the built-in ***AI_main()*** a try. Currently the starter code provides the initialization step which determines what blob corresponds to your bot, which blob corresponds to the opponent, and which corresponds to the ball.

In order to start the processing by ***AI_main()*** press '***t***' on the graphical window. This toggles A.I. processing on/off. When toggled on, the main loop calls ***AI_main()*** after each frame.

When the code is initialized, the A.I. state is set to zero. When in state zero, *AI_main()* performs agent identification – that is, it determines which of the blobs in the playfield corresponds to which agent, and what agents are actually visible (e.g. it may be there is no player 2, or that the ball is not there).

After pressing '***t***' you will see your bot move slowly forward for a short time, and the graphical window should show boxes for the detected agents.

You will also see a heading vector for your bot and possibly the other agents.

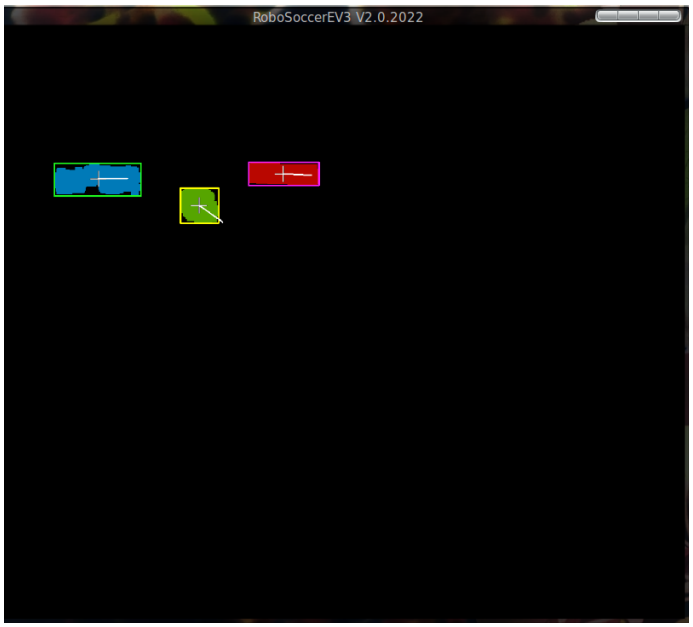


Image showing tracking of the bots and ball.

A cross-hair marks the estimated location Of the bots (after perspective offset correction) and ball.

The white line is the estimated direction vector (always aligned with your bot's uniform's long side)

The yellow line is the estimated heading vector.

You will need to learn how these change, and how/when to use them.

Notes on what the starter code gives you

Besides the image processing, blob extraction, blob tracking, and A.I. initialization, the user interface provides a ***manual override*** for the ***EV3***.

At any point when the A.I. is commanding the bot to do the wrong thing press 't' to toggle the A.I. off, followed by 'o' which sends an ALL_STOP command to the EV3.

Pressing '***q***' to end the roboSoccer program also sends an ***ALL_STOP*** to the bot.

During normal operation, the following EV3 commands are available from the keyboard:

'***i***' - Forward drive (toggle on/off)
'***k***' - Reverse drive (toggle on/off)
'***j***' - Spin left (toggle on/off)
'***l***' – Spin right (toggle on/off)

Use these controls to re-position your bot. However, note these controls won't work well while the A.I. is active (since your keyboard commands will compete with the A.I.'s commands for the bot's attention).

Note that unexpected program termination (e.g. segfault) will leave the bot in the last commanded state. If the bot is moving, it will keep moving until commanded to stop or powered off. ***Be ready to stop the EV3 manually to keep it from damage.***

Important note on vectors and quantities estimated by the image processing code

* ***Mind the velocity, direction, and heading vectors.***
  Each is informative, and each has shortcomings. You are supposed to use the three of them in tandem to figure out what is going on with each bot and the ball.

* ***All quantities are noisy***. The image processing pipeline is very noisy as a result of the webcam's low image quality. Blobs will change size and shape between frames, which in turn leads to noise in the estimates for all parameters (positions, velocities, headings). There is also the uncertainty caused by bots not being flat.

  Be sure to use the experience you gained during the Lander project to write code that includes noise management and that is robust to errors and imprecisions in the input.

That is all for the existing starter code. Let's get to your work!

Your Tasks for this Project

The general tasks you must solve for this project are:

1.- Building an EV3 socceer bot. ***[up to 10 % of project mark]***

Here, you do not want to build a big/fancy bot. It's going to be a competition so think small/fast/accurate.

Having a very fancy design will not earn you as many bonus marks here as having fancy software for the soccer playing part! Don't spend too much time building your bot. Your bot should be able to wear either uniform on short notice.

2.- Create a logo for your robo soccer team. [up to 10% of project mark]

Make a nice illustration for your bot's team. A particularly nice logo will earn you bonus marks!

3.- Design the state-based AI for the 3 game modes. [15% of project mark]

The game modes your code will support are:

0 – Standard soccer playing bot. The bot will play against an opponent and try to score more goals.

1 – Penalty shot. The bot will be placed somewhere on the field, the ball will be placed at the center of the field, and the bot must score a goal.

2 – Chase the ball. The bot continuously goes to whatever location the ball is at.

Each mode will have its own set of states:

For mode 0 -> states 1 to 99
For mode 1 -> states 101 to 199
For mode 2 -> states 201 to 299

The A.I. initialization step will leave the A.I. in state 1, 101, or 201 depending on the game mode selected by the user.

Think carefully about:
- what each mode must do
- what states it should contain
- what conditions (playfield configurations) will cause the A.I. to go to each state
- what robot actions will be needed for each state

Your Tasks for this Project

3.- Design the state-based AI for the 3 game modes (cont.)

Be precise. It won't be very helpful when coding to have states with very general names such as 'play soccer'. Ideally, each state should identify a single, simple behaviour that can translate into one function in the code.

Create a diagram showing the A.I. states and transitions for each game mode. This needs to be handed-in at the end of the project along with your bot's logo.

The diagram should indicate what conditions cause each transition, and also note within each state what functions in the code are associated with the state.

4.- Implement the state-based A.I. for penalty kicks. [20 % of mark]

Add code to ***roboAI.c*** enabling your bot to carry out a penalty shot. The bot will be placed at some location within its own side of the field, it then must make its way to the ball and kick it toward the opposing side's goal.

Speed is not the issue. Careful alignment counts. Power is not an issue either.

This: http://www.youtube.com/watch?v=uki7MikTLWY
is much better than this: http://www.youtube.com/watch?v=KZBOSEBOry4

Think carefully about how to make the behaviours that comprise this A.I. Mode general enough that you can reuse them for the other two modes!

In particular, ***do not hardcode the ball location at the center of the field, your code should be able to kick the ball even if it is placed elsewhere.***

5.- Implement the state-based A.I. for chasing the ball. [10% of mark]

Add to ***roboAI.c*** the code needed to implement the ability to chase the ball around the field and kick it. Upon initialization, the bot should proceed to wherever the ball is and kick it. After kicking, it should then proceed to the new ball location and kick the ball again. This will continue unil the A.I. is toggled off.

You should be able to reuse most of the code from ***step 4***. You may have to add code to handle field boundaries (more on that later).

Your Tasks for this Project

6.- Implement the complete A.I. for playing soccer against an opponent.
[35% of project mark for a fully working implementation]
[20% of BONUS project mark awarded competitively]

This will be challenging because you will not have a second (opponent) bot to test with. You must somehow come up with a suitable set of behaviours that will allow your bot to handle the presence of an opponent who is trying to score against you.

Think about:

When to attack (chase and kick the ball)
- How to optimize your path to the ball, and the kick direction

When to defend and where to place your bot
- Note it is illegal to park your bot inside your goal

When to back off (when two bots are in contention)
- This is robot soccer, not robot wars

All of these factors should be reflected somehow in your state-based A.I. for this mode.

You should be able to reuse your code for the other two game modes.

Here's where you get to show us how crunchy your bot can be. There will be bonus marks for very clever behaviour, for precise driving/kicking, for good strategy, and for being a good sports-bot!

Constraints and Game Rules

1) Your robot must not leave the playing field. The actual game field has a raised border, so at best your robot will push it around, and at worst it will get stuck on the border. Make sure your bot does not rely on going out the visible field.

2) Your bot must not charge/crash/kick the opponent. As mentioned above, we are not playing robot wars. Aggressive behaviour will be penalized, and if continued, will lead to your bot being disqualified.

3) Your bot can not park itself at the goal. You can defend by putting your bot between the ball and the goal, but you can not park at the goal. If your bot stays parked inside or in front of the goal for more than 10 seconds, it will be penalized.

Your Tasks for this Project

Constraints and Game Rules (cont.)

4) Your bot can not hog the ball. Once your bot reaches the ball, it must push it or kick it. The bot is not allowed to grab, capture, or drive the ball across the field. You should use this knowledge to determine whether you should attack or defend given the positions of your bot, the ball, and the opponent.

5) You can not intervene using the remote control. Other than to prevent damage to the bot(s). That is, if you notice your bot and the opponent are both trying to attack the ball and charging (unintentionally) into each other, you are allowed to use the keyboard controls to back off. However, the A.I. must not be toggled off, and you must not use the keyboard controls other than for the purpose of preventing problems.

That's All Folks!

The rest is up to you. Plan carefully, work consistently, and you will have a cruncy soccer-playing robot to show at the end of the term!

But wait! There's help!

Consulting: I will be available for consulting in person, times to be announced in the course forum. And as you know, I have an open door policy so drop by my office for any project-related issues or if you want input or advice on your design decisions. We can go to the lab if needed.

Tutorials will be devoted to consulting for the rest of the term, drop by as needed to test, request help from your TAs, and work on your project.

By email: As usual, email me directly with questions and bug reports.

Hints and Advice

- Think carefully how to implement kicking (if you have decided to implement a kicking mechanism). The kicking action must be carefully timed to avoid damaging EV3 components.

- Do not use absolute measurements. The size of your testing field, and the size of the playfield we will use for the competition will likely be different. Navigate using vectors, directions, angles, and relative distances.

- Use the EV3's sensors to your advantage. Clever integration of ob-board sensors to help plan and carry out actions will be prized at evaluation time.

- Test under different illumination conditions!

- Use the graphical display to plot things that help you see what your bot is doing. The starter code provides functions to plot on the graphical user interface. See how they work, and use them to help in your testing.

- Be creative when testing how your bot can play against an opponent. You do have two uniforms!

- Take some time to reflect on what you have learned through the course. What you are doing is not easy or trivial. It requires knowledge and a good amount of crunchiness. So feel good about your work!

Remember, even the best have trouble with robots:

https://www.youtube.com/watch?v=g0TaYhjpOfo

- Give us a pretty good bot! We'll be grateful and generous with marking!

Project Timeline

Thursday, Nov. 6 – First Progress check. By this time your team:
* Must have read the handout carefully
* Compiled and tested the starter code
* Completed the construction of your EV3 set
* Completed a diagram for the penalty kick A.I.

(nothing has to be implemented in code by this time)

Thursday, Nov. 13 – Second progress check (during lab). By this time your team:
* Must have completed the diagrams for the A.I. for all modes.
* Team Logo
* Must demonstrate working penalty kicking
* Must demonstrate working chase the ball behaviour

(FSM, logo, robot design and penalty kick will be ***graded***)

Thursday, Nov. 20 – Third progress check (in lab). By this time your team:
* Must show a working soccer-playing A.I.

Your robot must have working code to play ball against an opponent. We will have *friendly matches* at the lab for this. The robot must demonstrate ***going for the ball and kicking, defending, and chasing the ball.*** It should respond to actions by the opponent.
(Soccer playing A.I. will be ***graded*).**

Thursday, Nov. 27 – Make up for lost time – and friendly 1 on 1 games

Thursday, Nov. 27 to Monday, Dec. 1 – Fine tune your robot, strategy, and A.I.

Monday, Dec. 1 – Project due date – all code to be submitted electronically.

Tuesday, Dec. 2 OR Wednesday, Dec. 3 (depending on when I can secure the lab or a room for the required time slot)
(will post final date to Piazza soon as it's set)

Robo-Soccer competition. 9am to 3pm.
Pizza will be provided (we may have to eat outside)!

Tues. Dec. 3 – Wed. Dec. 4 – Return your EV3 kit/equipment

What to Hand In

Create a single, compressed archive caller ***UTSC_roboSoccer_teamName.tgz*** which contains your entire ***./utsc-robo-soccer/*** directory.

Your submission ***must also contain your team's logo, and your FSM design***.

Submit this compressed archive electronically on Quercus by the deadline

Understand this very important point:

*** *I will use the Quercus code to pre-compile and set up the directories for each team – therefore, whatever you submitted to Quercus is what you will be running on competition day.***

One final reminder:

Please study well for the final! You worked hard all term, don't let your guard down now!