Final Project – NXT Robo Soccer

This is it! Time to take everything you have learned and put it into use to create the mightiest NXT 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:

You will implement a soccer-playing robot able to participate in a one-on-one competition for most goals scored.

You will apply your knowledge of sensors, sensor noise, and noise management in order to provide your bot with reliable data for planning and action execution.

You will write software to handle a dynamic environment, with an opposing agent whose behaviour cannot be predicted in advance.

You will gain practical experience working with vision-based systems, and understand their limitations and potential pitfalls.

You will understand simple, state-based AIs capable of providing robots with a suitable set of behaviours to handle specific tasks.

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.

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 NXT bot for most of the testing and development. Therefore, you're strongly advised to work at the IC 406 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 cool!

This project provides you with an opportunity to bring together all the tools you have acquired during the term, strenghten 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:

All the infrastructure for the project was developed during the summer of 2013 by

Kevin Lee – Bluetooth integration, AI testing
Per Parker – Systems integration, build chain, AI scaffold
David Szeto – Robot control API, AI and system testing
Paco Estrada – Image Processing, AI design

Project Overview

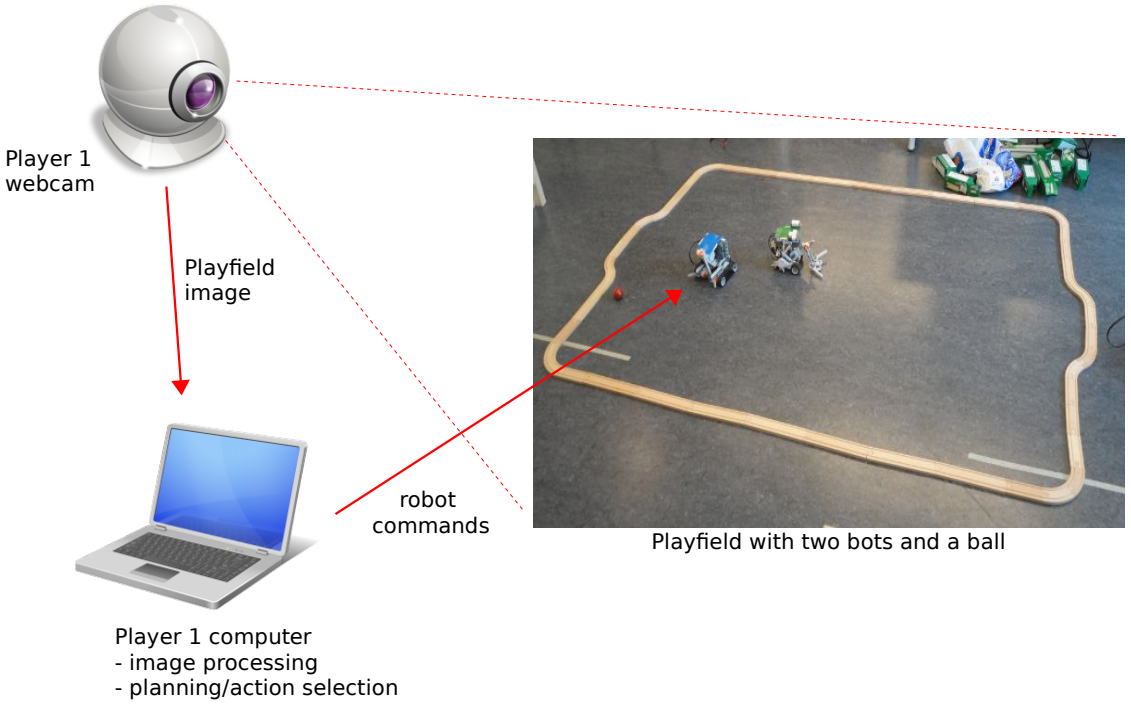
As you have learned from the localization project, the NXT platform is somewhat limited in what you can do with the sensors and CPU/memory on board the intelligent block.

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. The NXT's colour sensor is sadly inadequate for this, and the intelligent block itself lacks sufficient memory and processing power to be able to handle a camera, or to process information fast enough to play soccer at a reasonable pace.

Therefore, for this project, we will work with a multi-component system.

- The NXT bot will be remotely controlled (via its bluetooth channel)
- A laptop/desktop computer equipped with a webcam will do all the work of processing the soccer field image, planning actions, and sending appropriate commands to the bot

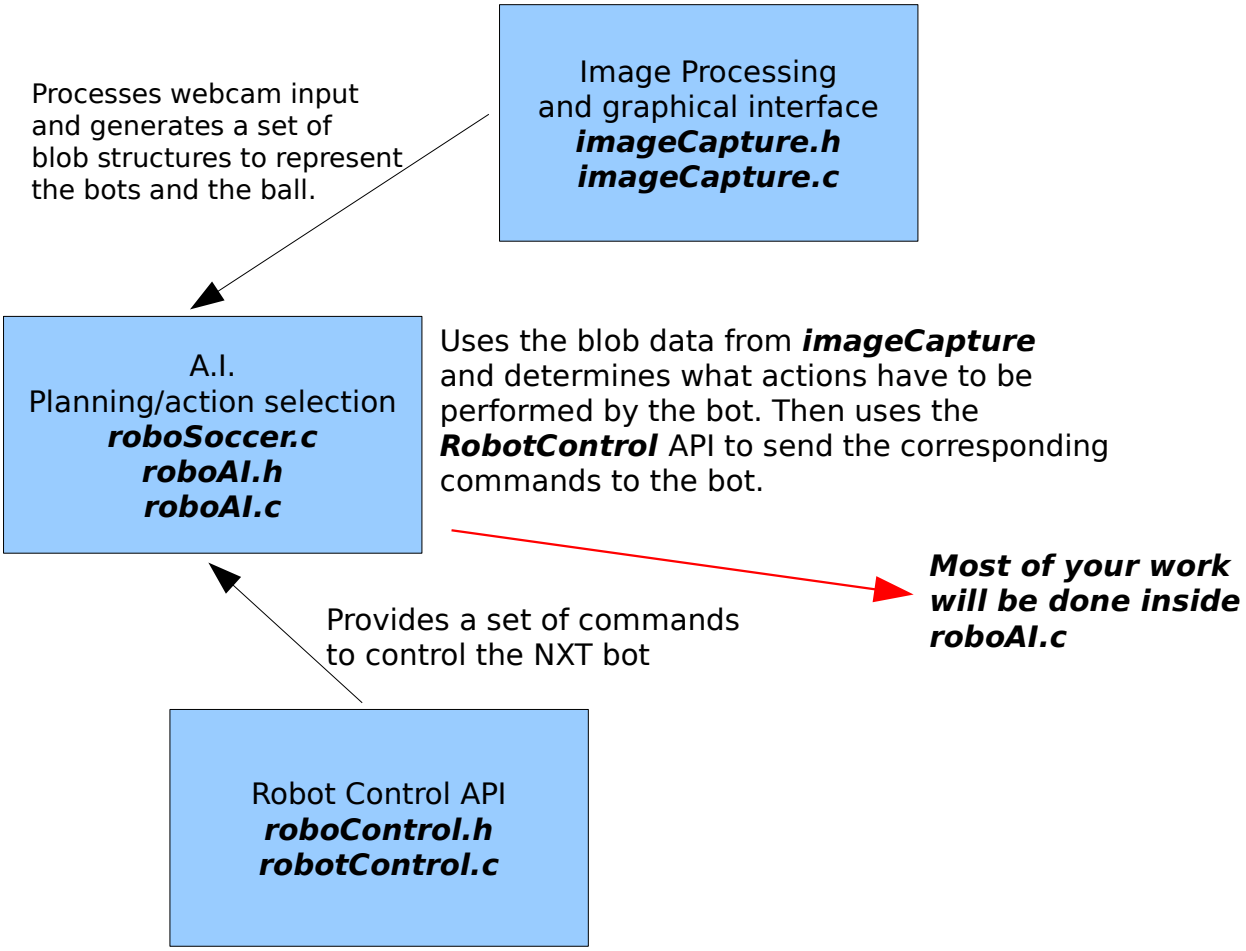
Below is an overview of the entire system from the point of view of player 1:



Similarly, player 0 will have a webcam and computer. The NXT bot here acts as a remotely controlled drone. All the processing is done by the computer.

Structure of the Software Distribution

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



What you need to read and understand:

imageCapture.h: You must understand what the ***blob*** data structure contains so you can use it.

robotControl.h and the ***README.txt*** in *src/API/***:** These are the controls you have at your disposal for controlling the NXT bot.

RoboSoccer.c: This is the program entry point. ***You will need to change the NXT hex ID at the top of this file to match your NXT's bluetooth ID.***

What you will need to read, understand, and expand:

roboAI.h, roboAI.c: Here's where you will implement your soccer playing code!

Setting things up

1) Set up your play field

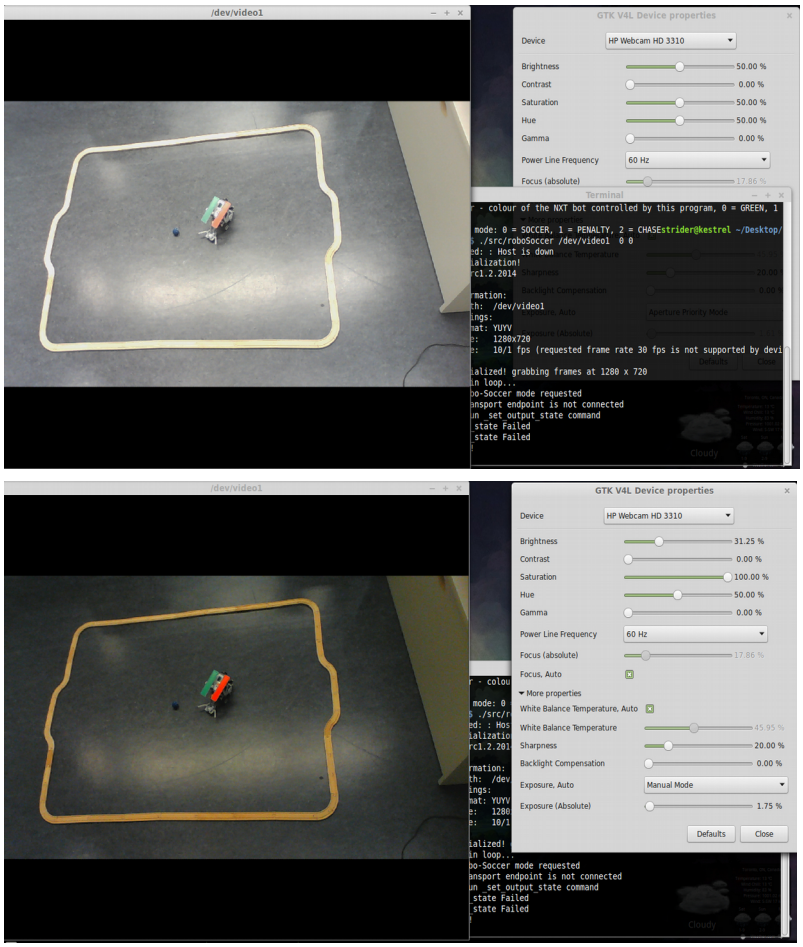
You will need a rectangular playing field. Set it up on a neutral coloured surface. Ideally use a gray surface similar to the colour of the classroom floors and lab at IC. This is where your bot will play during competition day.

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.

Call up '***gtk-v4l***' to set up the camera's controls so you get a clean view of the field.



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. Make sure to adjust the exposure manually, and ***boost saturation to 100%***.

Do not close gtk-v4l. Minimize it and keep it around so the camera doesn't go back to auto-adjust, or in case you need to tweak parameters during play.

To reset the webcam, unplug it and plug it back in.

Understanding the Information Flow

Compile the code and make sure there are no errors (***you may have to re-compile the nxtlibc-0.1 library*** in *utsc-robo-soccer/Bluetooth/).* First thing for you to do is run the code and understand how to do the initial set up. From the command line, run:

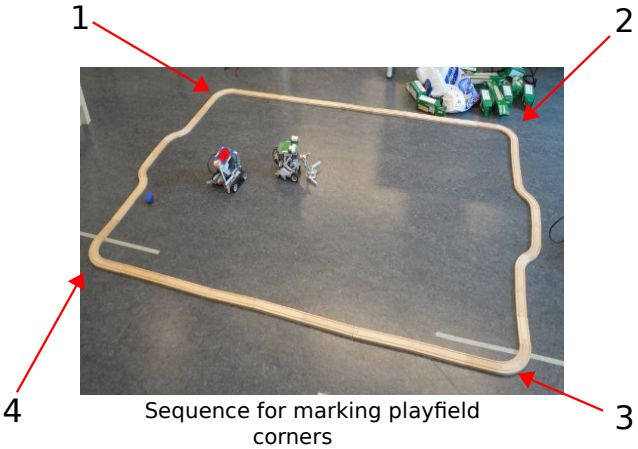
>> ./src/roboSoccer /dev/video1 0 0

Here '*/dev/video1*' is the device identifier for the USB webcam. On some computers this may have to be changed to '*/dev/video0*'. If you get an error starting up, check the /dev/ directory to find out the video device ID on your computer.

The command above will open a graphical user interface window (no buttons!) showing the current image from the webcam. ***Make sure the webcam can see the entire playfield***.

Make sure the playfield is empty!

You now have to provide the locations of the playfield corners in the order shown below:



Sequence for marking playfield corners

Press '***m***' on the graphical window to start the corner capture process.

This brings up a green cross, use ***WASD*** and ***Shift + WASD*** to move the cursor to the corner location to be marked next. Press ***space*** to mark a corner. The corner position will be printed on the console.

Once the four corners are marked, the image will turn black. At this point the playfield initialization is complete. ***The program caches the corner calibration parameters.*** Once registered, you can skip this step by pressing '***g***' to load the cached values. This will work as long as you do not move the camera, change the shape or size of the field, or change the lighting in the room.

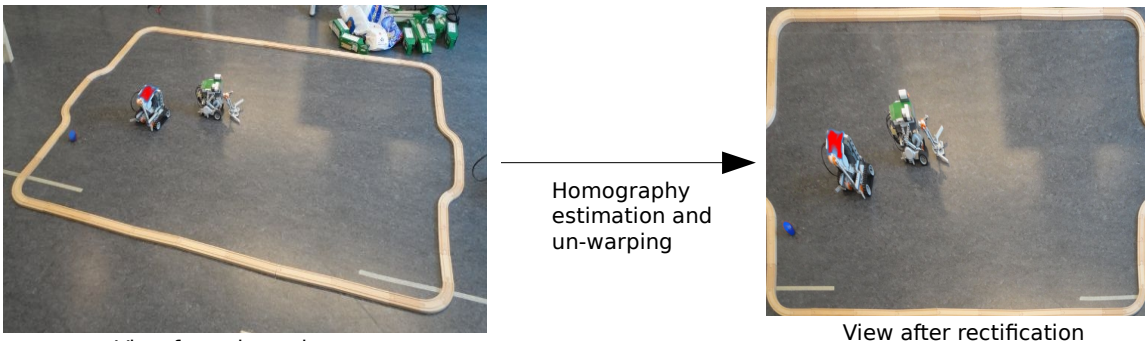
At this point, the code is processing input from the webcam and looking for the bots and the ball. The bots and ball are identified by unique colours. One bot is ***green***, one is ***red***, and the ball is ***blue***. This is why having a neutral-colour background helps.

Place your bot and the ball on the playfield at this time. If you have not yet built your bot, place its '***uniform***' as a marker somewhere to test the set-up and image processing initialization code.

Understanding the Information Flow

Why do we need to mark the four corners of the field?

*The corner locations are used by the code to **rectify** the image of the playfield. That is, the code converts the image of the playfield into a rectangle so that distance and position estimates are easier to make.*



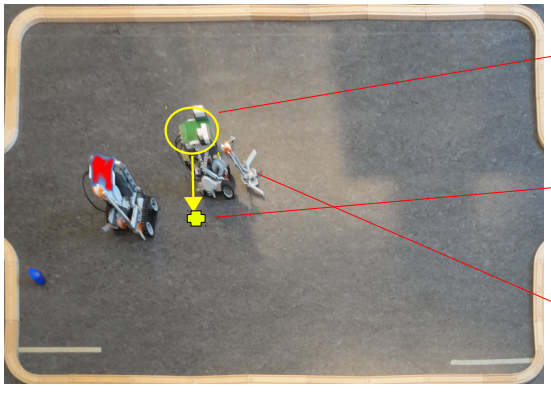
View from the webcam

View after rectification

The image rectification process employs a ***2D Planar Homography*** to un-do the perspective distortion from the camera. See

http://www.cs.toronto.edu/~jepson/csc2503/tutorials/homography.pdf

to learn how this works. Note that the rectification process ***distorts*** objects not at the surface of the playfield. The images of the bots are distorted in the example above. This means that ***there will be an unavoidable error when estimating the position of the bots.*** If we wanted to have much more accurate position estimates, we would have to either ***use 2 cameras and do 3D estimation***, ***have a model for the robot***, or ***use a device with a built-in range finder*** such as a Kinect sensor.



1) Robot location is reported for the robot's coloured body

2) However, the robot's position on the field is further down the image. Our code can't estimate that since we don't know what your robot's height is.

3) Further, the position of the 'leg' is away from the robot's body or its location on the ground.

Uncertainty in bot location
As a result of using a single
Image and no robot model.

You need to perform height offset calibration to correct the position estimate. Read on.

Understanding the Information Flow

Image processing pipeline – What you see is what you get

It is important to understand exactly what the image processing code gives you. Here is the image processing pipeline after the initialization is complete:

1) Image capture: obtains a single frame from the webcam. Resolution is 1280x720.

2) Image rectification. Results in a rectangular, perspective-corrected field 1024x768 pixels in size.

3) Background subtraction and colour pixel detection. During initialization, a picture of the empty field is captured and used for background subtraction. Each pixel is compared against this reference. Pixels similar to background are set to black. Pixels for colorful objects (hopefully the bots and the ball) are preserved.

   The goal of this step is to provide contiguous colour regions for the two bots and the ball. Because you are working with a webcam, environmental lighting conditions will have a strong effect on the results of this step. For this reason, the code provides adjustable controls for the main parameters governing colour-blob detection.

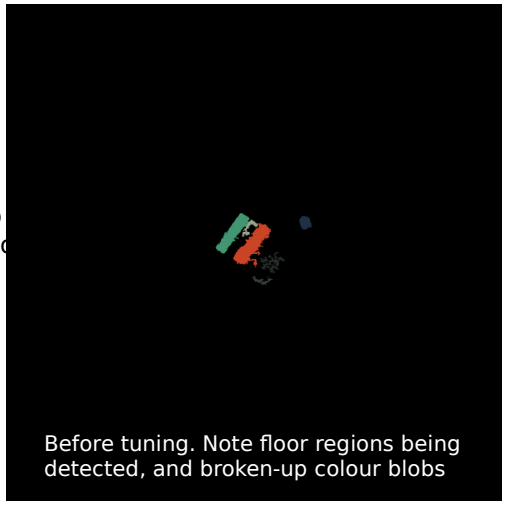
   On the image window:

   Use '[', and ']' to decrease/increase the colour detection level. Increase it if the code is detecting a lot of pixels outside the coloured regions of the bots and ball.

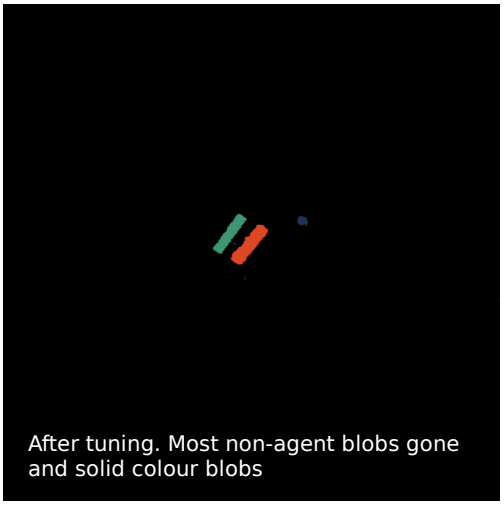
   Use '{' and '}' to decrease/increase the colour angle threshold. Decrease this value if the code is breaking the colour regions into many parts (e.g. the green bot's uniform results in multiple regions with slightly different shades of green).

This tuning must be performed each time, as lighting changes over time.

The end result of the image processing step will be a list of coloured blobs. Each blob, if all works well, will correspond to one of the three agents: green bot, red bot, and blue ball.



Before tuning. Note floor regions being detected, and broken-up colour blobs



After tuning. Most non-agent blobs gone and solid colour blobs

Calibrating the height offset adjustment

In order to get a reasonable position estimate for the bot positions, you must provide two calibration points for the program to use to estimate the hieght adjustment needed anywhere else in the field.

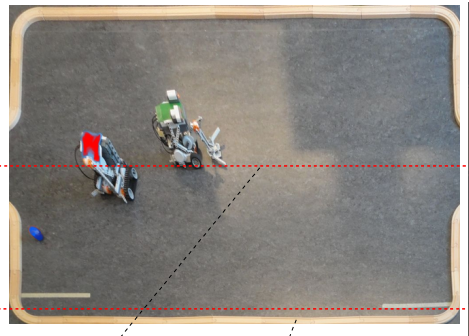
Calibration should be performed for both bots at the same time prior to competition. We can not use calibration data from your robot for the opponent since the height of the robots will be different.

1) Place both robots midway between the top and bottom of the field. Make sure the center of your bot's uniform is aligned with the middle of both goals.

- Press '***z***' to start calibration (bots will be identified and tracked)
- Press '***x***' to record the first calibration location

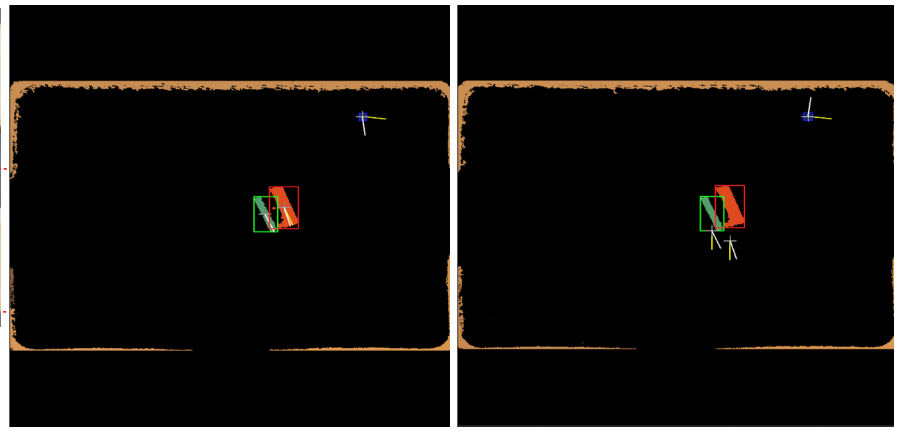
2) Place both bots as close to the bottom of the field as possible

- Press '***x***' to record the second calibration point
- Calibration data is now available and the code will cache it. Thereafter you can re-load it by pressing '***c***' without the need to go through this process again.



Bots should be centered along this line for the first calibration step.

Bots should be located along this line for the second calibration step



Bot tracking before (left) and after (right) calibration. Notice the position of the bot has shifted down to account for the correction pf the perspective projection error.

You're now all set to go! If you need to restart the program for any reason you can now just press '***g***' then '***c***' to re-load cached field rectification and perspective correction data.

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 are not supposed to go into the image processing code other than to learn how it works.*** 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. 0 for the ball, 1 for the green bot, 2 for the blue bot, anything else is not an agent in the game.

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 structur is all you need to plan your strategy. Make sure you use all of it smartly.

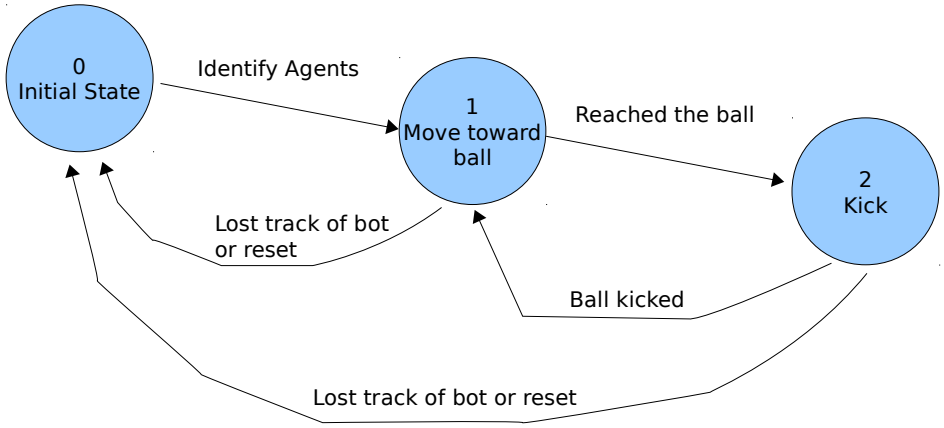
Understanding the Information Flow

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.

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 green or blue 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 ***README.txt*** in ***src/API/***

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

1) On the NXT, go to Bluetooth options, search for nearby devices, select the computer you want to pair the both with and start the pairing process.

   A pairing code will appear on the screen of the NXT. Input that code on the computer's pairing request box to complete the pairing.

2) Note the NXT'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. If when you run the code you see the following error:

```
write: : Transport endpoint is not connected
Failed to run _set_output_state command
_set_output_state Failed
_set_output_state Failed
```

   It means the code 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 command 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.

* Green box around the bot controlled by this program
* Red box around the opponent bot
* Blue box around the ball

The box colours are independent of the bot colour!

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

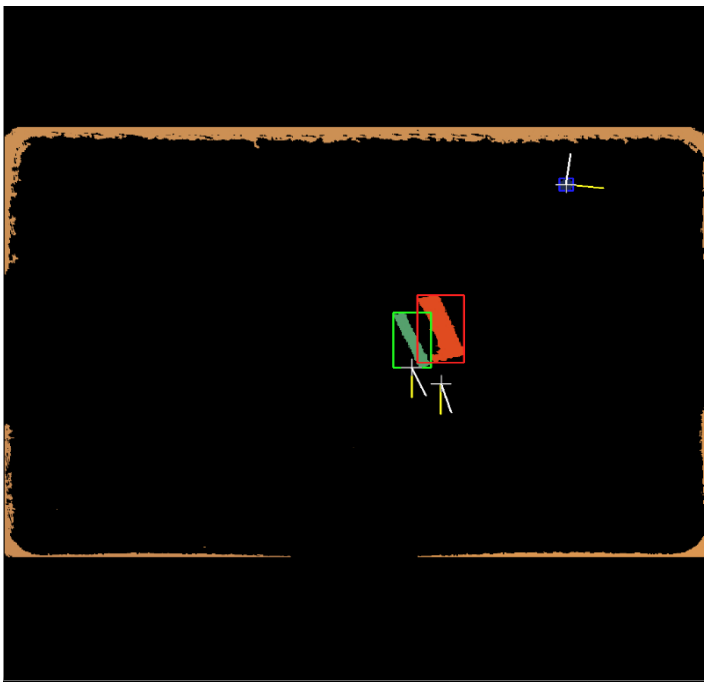


Image showing tracking of the bots and ball.

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.

Notice the heading and direction do not agree. You must figure out when and how to use each.

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 ***NXT***.

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 NXT.

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

During normal operation, the following NXT 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 NXT 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 a soccer NXT bot. [up to 10 marks]

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 marks]

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.

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 and kicks it.

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.

The A.I. designs and bot logo are worth [20 marks] (not counting bonus for cool logos!)

4.- Implement the state-based A.I. for penalty kicks. [20 marks]

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 marks]

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. [25 marks for completing this, plus up to 25 marks depending on how well your bot plays during the competition]

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 every Friday from 11am to 1pm. 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 NXT 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.

- Legend has it that the NXT can be running its own code while responding to Bluetooth commands. The NXT has its own sensors... those may be useful...

- Test under different illumination conditions!

- 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!

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

Project Timeline

Monday/Tuesday, Nov. 3-4 – Pick up your NXT and soccer kit at the end of lab
Start working on your project. Read the handout and make sure you understand what you need to do for the first progress check.

Tuesday, Nov. 10 – 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 NXT set
* Completed a diagram for the penalty kick A.I. (nothing has to be implemented in code by this time)

Wed/Thu, Nov. 11-12 – Consulting sessions with your TAs. Drop by and ask questions

Tuesday, Nov. 17 – Second progress check. 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

Wed/Thu, Nov. 18-19 – Consulting sessions with your TAs.

Tuesday, Nov. 24 – Last progress check. 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.

Wed/Thu, Nov. 25-26 – Consulting sessions with your TAs.

Tuesday, Dec. 1 – Competition warm-up. Teams should have fully working robots at this time. Friendly matches in the lab – fine tune your strategy for the competition!

Thursday, Dec. 3 – Project 4 due date.

Monday, Dec. 7 – Robo-Soccer competition. 9am to 3pm. Pizza will be provided!

Tuesday, Dec. 8 to Wednesday, Dec. 9 – Return your NXT kit/equipment

*** Schedules for progress checks will be done by Doodle poll, as usual.***

What to Hand In

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

Submit this compressed archive electronically on mathlab by ***9am on Nov. 30, 2014.***

```
submit –c cscc85f15 –a P3 –f UTSC_roboSoccer_teamName.tgz
```

Submit your state-based A.I. diagrams and your team logo either in electronic form by email to the instructor, or in written form at the C85 drop-box by the above deadline.

Once you have done that, celebrate! You've completed all the work for C85!

Please study well for the final! I don't want anyone to get a surprise there!