System Design Project 2012 – Group 8

Milestone 1 Individual Report
Gediminas Liktaras (s0905195)

February 2, 2012

Goals

Up to the first milestone I worked primarily on the vision component of the robot system. Since three people were working on this component and my other two teammates were engaged in recognition portion of the system, I set upon myself the task of creating the communication infrastructure of the component. This included the way the vision subsystem would represent and make available the state of the football field, as well as the way it could be adjusted from the GUI window.

Vision Subsystem

The task of the vision subsystem is to capture the overhead camera feed, extract the current state of the football field and make this data available to the rest of the system.

At first I have connected the application to the camera feed. This was a fairly easy process, since there was a lot of working code to take examples from from the previous year. Our application uses v4l4j library to implement this.

Then I directed my full attention at the vision subsystem's communication layer. This choice was motivated both by the fact the other two people in the vision team were working on the recognition portion of the subsystem and because a functional vision system was a hard requirement only for the second milestone.

Creating the world representation was a fairly easy task, which was accomplished by a few minimalistic container classes. The real challenge was the presentation of field state information to the rest of the system. The resulting system had to be easy to use, flexible in the sense that any number of components may require access to the visual system and asynchronous, so that the vision thread can spend all its resources on world state extraction.

I eventually solved this problem and my solution not only satisfied all the above criteria, but it also fully encapsulated the vision system and provided a nice layer of abstraction to make seamless world state provider replacement by the simulator. I have implemented the communication channel with the help of the Observer design pattern and Java's thread synchronisation features.

Each component that wants access to world state updates would create an observer object and connect it to the main vision system observable. Each time fresh data would be available, the vision system would notify all observers with the new world state and they would then pass this data to the "client" parts of the system.

I have taken special care to provide safe asynchronous access to world state. Each observer is independent, so different vision system clients can consume the data at any rate they are comfortable. Furthermore, the observers use intristic locks and the built-in wait-notify mechanism to make this data exchange asynchronous and thread-safe.

I have also implemented a method to customise the vision subsystem via GUI. It was a much simpler solution – I simply put all image recognition portion settings into a separate class and exposed it via the main vision class. This way the rest of the system can get and set the configuration of the image processor.

GUI

As a side effect of the implementation of the vision system customisation logic, I have created a basic skeleton GUI. At the moment of writing it contains the (annotated) display of the field and some basic vision controls on the side. The team has agreed to use it as a basis for the rest of the GUI implementation.

I have used the WindowBuilder Pro Eclipse plugin for the GUI's creation. It greatly simplified the creation process via its drag and drop workflow.

Future Work

I plan to finish whatever work is left on the communication component and invest all my effort into world state extraction. Although it will take some experimentation and testing, I hope to have the vision subsystem online by the next milestone. Further goals will be decided then.