Project Overview
The RP10 Platform projects have their origin in the Kate Gleason College of Engineering (KGCOE). A series of multidisciplinary senior design projects in KGCOE, most recently P08201, have culminated in the second generation of the RP10 robot – the 10 referring to the maximum payload of 10 kilograms. Other robots in development include the smaller RP1 and the larger RP100 and RP1000 (the latter two in design phases).

The System Software project team is tasked with creating a system that will use a microcontroller with embedded software to communicate with a PC and execute navigation instructions on the RP10. A library will be built to provide an API for PC software to interact with the RP10 through multiple programming languages. Both a graphical and text-based control application will be built on top of the API to allow a user to manually control the robot. The initial focus for the project is on simple navigation and wired communication, enabling the users to easily interact and move the robot around. The initial plan is to support movement with 1 to 4 motors and to allow the robot to adapt to any motor additions or malfunctions.

The Model team is tasked with creating a three-dimensional full-scale simulation of the same RP10 robot. It will implement all the existing features of the RP10 using Microsoft Robotics Development Studio (MRDS). Engineers will be able to create programs with Visual Programming Language (VPL) in MRDS to command the simulation. When satisfied with the results of a program after testing against the simulation, they will be able to run the program against the actual RP10, using the System Software team’s API behind the scenes without changes to their program. Additionally, the Model team will also deliver a platform characteristics document describing the physical characteristics and capabilities of the RP10 for both documentation purposes and to definitively establish the simulation parameters; obviously, the simulation must match the RP10’s abilities (within acceptable bounds) in order to be valuable.

In summary, the final deliverables include the source code of platform and model software, design documentation, the Simulink model describing platform characteristics, user manuals for installation and usage of software, and a demonstration of the simulation and control applications. All deliverables will be deployed to the KGCOE’s EDGE content organization website. The demonstration will be provided during the final project presentations occurring in Weeks 9 and 10 of Spring quarter (20083).
Basic Requirements

Platform

- The platform software can send to and receive commands from the MCU via a serial cable connection.
- The RP10 platform software can support control of 1 to 4 motor modules.
- The RP10 control software can control the platform in accordance to the electronic bay configurations.
- The proportional power of each installed drive motor can be set to an integer between 1 and 100 percent to sustain motion until stopped.
- The proportional power of each installed steer motor can be set to an integer between 1 and 100 percent to sustain motion until stopped.

Linear motion is governed via bays, explained later in this document. Rotational direction is governed via bay as well, but the power to each rotational motor can be controlled individually. All motors will be powered on a percentage basis from 1 (minimum) to 100 (maximum duty cycle and therefore power).

- Each installed drive motor can be set to spin either in positive or negative direction relative to the wheel rotation axis.
- Each installed steer motor can be set to spin either in positive or negative direction relative to the wheel joint rotation axis.

The direction of motion is ultimately vague by definition, as the only difference between forward and backwards from a user standpoint is the initial orientation of the motors and therefore can only be defined as positive and negative from a design standpoint. As such, each motor, both rotational and linear, must be able to move in either a positive or a negative direction.

- Each installed drive motor can be individually stopped by setting power to zero.
- Each installed steer motor can be individually stopped by setting power to zero.
- The platform can be emergency-stopped, reducing the power of all motors to zero.
- The MCU software can read the voltage of the robot’s batteries.
- The API supports retrieval of the current battery level on the platform in volts.
- The API provides the maximum battery level on the platform in volts.
- The control application displays the current level of battery power.
- The MCU software can read the value of each encoder.
- The API supports retrieval of each encoder’s current value.

The RP10 as provided is not equipped with fully functional encoders, nor does the MCU have the capability to fully utilize all encoders (if all 4 motor modules are installed). Further upgrades of the platform itself may yet provide this functionality. It is with this in mind that the MCU
must be capable of reading all encoders and the API must provide support for retrieving that encoder data.

- If the RP10 MCU does not receive a heartbeat message from the PC in a specified time frame, the MCU will terminate any ongoing operations and turn off.
- The MCU software will read external input ports.
- The API allows retrieval of the values of external input ports.
- The MCU software will write to the external output ports.
- The API allows one to set the external ports' output values.

All interactions with the robot (as described above) will be performed through the text-based or graphical control applications. All described commands will be available in both UIs. The control application and any other API-dependent software will require Windows XP or higher with .NET Framework 3.0 or higher. Visual Studio 2008 is recommended for further development on this software.

**Model**

A full-scale three-dimensional model of the RP10 platform must be delivered. This is a dual requirement: both the visual aspect (the recognizable appearance of the RP10) and the functional aspect (the capabilities and physical actions of the RP10) must be present. This functional aspect includes its physical characteristics (mass, acceleration, etc.) and all functionality mentioned above in the Software System requirements, outside of heartbeat and voltage indicators – the simulation should assume perfect communication and battery life.

The simulation must be capable of being driven by a program written using MRDS’s Visual Programming Language. This same VPL program must also be able to drive the RP10 itself with few to no modifications.

The Model team must also deliver a document characterizing the physical capabilities of the RP10. This includes the voltage requirements of the system; torque, acceleration, and speed of the individual motor modules; and the acceleration and speed of the system as a whole. This will be done using Simulink and will be geared towards creating a fully realistic simulation.

**Constraints**

The primary constraints on the System Software design are the MCU, means of communication, and the division of functionality. The MCU chosen by the team is the Freescale CSM12D 16-bit microcontroller unit. This controller was chosen for two reasons. First, this was the microcontroller used in P08201, the preceding project that produced the RP10. Second, the Computer Engineering department had these microcontrollers available for use. The communication, as discussed in the requirements above, must be through serial cable, the easiest means of communication with the robot. However, there are still the physical constraints that stem from the limited length of the cable. The team is forced to consider the feasible working range of the robot with this in mind from both a testing and deployment standpoint. A few design trade-offs must be considered concerning the division of functionality between the PC and MCU. The software must run on the Windows NT family of operating systems, the platform
of the customer and the target audience. Regarding the model, the simulation environment must be Microsoft Robotics Studio (MDRS).

**Development Process**

The team adopted was the Spiral process methodology. Numerous risks were realized at the outset of the project, and the Spiral methodology is built on risk-based planning. Due to the abbreviated lifetime of the project (6 months), it was decided that 2-week cycles were best. The first of these began in Week 4 of Winter Quarter (20082). The sponsor had no process requirements and approved the team’s choice.

Adherence to the methodology during Winter Quarter was shaky at best. Cycles were often either longer or shorter than planned due to a number of factors, and team progress was not tracked. This was addressed during the interim meeting with the team’s advisor, and the methodology was reapplied beginning in Week 1 of Spring Quarter. From this point to the end of the project lifetime, there was a far greater structure to the team’s application methodology, and the results showed. A risk management document was created and addressed during every weekly meeting with the advisor and sponsor, and project responsibilities that week were adjusted according to the identified risk impacts and likelihoods for that week.

Another process change for the Spring term was the ticketing system, built into the team’s Trac wiki yet not utilized in the first quarter. A series of tickets was created, each corresponding to a piece of work necessary for project completion. While each ticket was not weighted, team members were encouraged to break down tasks into as many parts as possible in order to show incremental progress. The improved work transparency allowed the team to identify areas of concern as the term moved on.

One thing the process did not do, however, was specify team roles. Outside of Sahil, the team leader, and Jeff, our note-taker, there were no solid overall team roles. The System Software team had originally consisted of Sahil, Karl, Joe, Adam, and Paul; the Model team was originally Aaron, Jeff, Kyle, and John. As it turned out, most of the team stayed roughly in the initial configuration with the exception of Sahil. At the end of the first quarter, he moved to the Model team as more roadblocks appeared but ultimately spent most of his time in overall organization of both sub-teams.

**Project Schedule: Planned and Actual**

The initial project schedule was developed during the first three weeks of Winter quarter. As with many early plans, it was abundantly optimistic and in no way reflected a full understanding of the project. For example, we desired to have partial motor control of the simulation as early as the end of the first cycle – a task that was incomplete almost four months later. Much of the early schedule development had a narrow focus that pertained to the current cycle alone with little forward thinking. This changed significantly during the second quarter; the team estimated tasks for the entire quarter, broken down in cycles, and adjusted expectations based on work done in that cycle.

Based off of the functional requirements, key milestones were set up early and ultimately remained unchanged, unlike the time frame in which the team expected to achieve them. For the System Software side, the early goals included reconstructing the previous Senior Design team’s work in order to get an understanding of how to do MCU programming, API design, and serial
communication (given the team’s relatively lack of expertise in those areas). Once those were understood to a greater degree, a series of desired motor functions was developed, as well as a design for a lightweight communication protocol. Once the commands were finalized, the API was built to deliver those commands using the communication protocol, and the UIs were built to give a user a simple interface for sending those commands.

From the Model side, early estimates assumed a SolidWorks model in MRDS within the first couple of cycles, as indications were promising that such a goal was achievable. Working alongside the System Software team as they developed the API, the plan was to know what commands needed to be implemented in MRDS in order to have a simulation with the same behavior as the RP10. Definition of the platform characteristics would occur in parallel with development of the model. Once the simulation was complete, a test application would be developed in VPL for testing and rework purposes to flesh out the final simulation.

The actual schedule, as it turned out, was quite different for all involved. As mentioned, portions of the Model team schedule were so drastically adjusted backwards, it is reasonable to call that portion of the project a failure. The SolidWorks/Blender visual model was completed and imported into MRDS successfully; however, the time frame for its completion was extended by months due to the extreme difficulty in acquiring the software necessary to do so. The platform characteristics document was completed on time, as well, and was one of the few things not pushed back (outside of the inability to test it due to the hardware failures discussed later).

Additionally, the simulation segment of the Model project was incomplete in the end. As seen above, the initial task seemed a simple matter – too simple to be true, unfortunately. Creation of the back-end services, necessary to drive the simulation, proved an insurmountable task. While portions of them were complete (specifically the entities that allowed the SolidWorks model to be viewed in MRDS), the overall architecture was in the end too dense to fully create a working simulation.

The System Software team had separate schedule difficulties. The biggest obstacle was the hardware. Time was burned in attempts to track down a number of electrical issues with both the RP10 and the microcontroller unit. Lacking any team members with extensive electrical engineering experience, it was up to the software engineers to attempt to track down the issues. To their credit, many of the issues were sufficiently narrowed down enough to conclusively say what was wrong with the robot; unfortunately, some issues were beyond their ability to repair. Aside from the fact that the robot didn’t work, the time spent hunting down stray electrical signals took away from software development time, leaving a product that, while functional, ultimately does not reflect what the team feels was potentially possible. On the other hand, the API and serial protocol were both functional on schedule.
System Design

Architecture

The following diagram presents the layered architecture, mixing software and hardware components (colored).

The topmost software layer is for applications that control the RP10. Text-based and graphical control applications are interactive front-ends to the RP10 application programming interface (API). For example, users can start and stop motors, as previously described in the requirements, by entering text commands in the text-based application or clicking buttons and dragging scrollbars in the graphical application.

The model project is represented in a simulation in Microsoft Robotics Developer Studio (MRDS). A Visual Programming Language (VPL) program commands the simulated robot, changing properties such as velocity and position in the simulated physics and 3-D visualization environments. When switching the target of the VPL application from simulation to the real platform, it sends commands to the RP10 through the API’s .NET binding.

The second software layer is the API, consisting of potential bindings in various programming languages. A binding for the Microsoft .NET framework was the focus for this project. It presents functions for commanding motion (i.e., turn, drive) and querying diagnostics (i.e., battery power). It translates function calls into messages sent from the PC to the RP10 via serial cable.
On the RP10 microcontroller, on-board platform software receives commands from the PC and translates them into lower level hardware commands to affect the RP10 components. Command acknowledgements and data (on diagnostic queries) are returned to the PC. The on-board software is entirely event-driven, sending commands to the PC only in response to previous commands.

**Quality Concerns**

Several quality concerns influenced the architectural decisions:

- The platform software must be extensible, allowing various types of applications to control the robot. Extensibility is achieved with separation of the API from its applications. The text-based control, graphical control, and simulation applications are built on top of the features of the API.

- As robot hardware can change (i.e., varying number of motors), the API must support means of commanding different hardware configurations. This flexibility is achieved with separation within the API, dividing between configurable control rules for the robot and a protocol for communicating with a robot. Rules include the configuration of a robot with motors, idle or active; motor bays, grouping motors so changes to one affect all grouped; and selection of a protocol. Each protocol represents a communication medium such as serial cable or USB.

- The reliability of robot communication is variable between potential wired and wireless mediums. Messages could be lost in transmission. To prevent the robot from moving continuously when a stop command has been lost, a heartbeat is applied. The PC sends heartbeat to the RP10 at a fixed frequency. If the RP10 does not receive a heartbeat within the expected time frame, it will stop any active motion.

- To support modifiability, the robot’s behavior is defined in the implementation of the API. Error checking is done almost entirely on the PC before transmissions. It is easier to modify and maintain the PC software than the proprietary hardware-specific on-board software.

**Rationale of Technology Choices**

Several technologies were chosen to implement various aspects of the architecture:

- The Microsoft .NET Framework was chosen as the first binding for the API. With numerous languages supported through its Common Language Runtime (CLR), one binding in one of the .NET languages would be accessible in others. The C# programming language was chosen because it is feature-rich and similar to the more common C++ and Java. The .NET Framework also supports rapid development with drag-and-drop UI creation and is the foundation for programming robot services in MRDS. The Windows Presentation Foundation was the basis for the graphical control application.
• On the surface, MRDS presented most features for comprehensive simulations, including a 3-D visualization of the simulation environment and an integrated physics engine. It also included VPL, a simple programming environment for novice programmers. It was also freely available through the Microsoft Developer Network Academic Alliance.

• A Freescale microcontroller was used in the P08201 project when the robot was constructed. Taking a path of least resistance, the team chose to take advantage of the previous team’s experience. The microcontroller was also available from the Department of Computer Engineering.

Process and Product Metrics
The team did not collect metrics during the first quarter of the project outside of the required time tracking. Several product metrics were discussed, including the standard errors per KLOC, but were determined to be useless as little progress was made on the product. During the second quarter, three metrics were put into place:

1) Slippage
2) Project Completion Percentage
3) Test Pass Percentage

In order to track slippage, the team required a means to track what tasks of a project cycle were complete or not. A list of all tasks required for full project completion was compiled into a series of tickets, assigned to the various team members in charge of those parts of the functionality. The tickets were then closed out when completed or broken down into more specific functional components. At the end of each cycle, the uncompleted tickets slated for that cycle were tallied and reported as the total slippage while being pushed into the next cycle.

Project completion percentage also built off the ticket system. This metric is simply the number of completed tickets divided by the total number of tickets.

The test pass percentage is also self-explanatory. A test suite was created to verify and validate the team’s software. The metric is the number of passed tests divided by the number of completed tests.

<table>
<thead>
<tr>
<th>Cycle</th>
<th>Slippage</th>
<th>Completion</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>5</td>
<td>53%</td>
</tr>
<tr>
<td>5</td>
<td>21</td>
<td>61%</td>
</tr>
<tr>
<td>6</td>
<td>5</td>
<td>74%</td>
</tr>
<tr>
<td>7</td>
<td>8</td>
<td>84%</td>
</tr>
</tbody>
</table>

A final total of 166 tickets were created by the end of the project. It’s apparent that steady progress was made during the second half of the project. One drawback of this metric is that all tasks are weighted essentially the same no matter their complexity – one reason the team was encouraged to break down tasks as much as possible. Note the following facts:
Cycle 4 started at 53% project completion because 1) it was the first cycle with progress tracking and 2) the previous quarter’s work was counted.

Cycle 5 planning severely overestimated the number of expected tasks, hence the slippage. These tickets were then spread out through the remaining cycles.

Much of the slippage in cycles 6 and 7 are attributed to continual hardware and simulation difficulties. The team kept expecting that the current issues would be resolved in the next cycle.

<table>
<thead>
<tr>
<th>Test</th>
<th>Total Tests</th>
<th>Failed</th>
<th>New Failed</th>
<th>Still Failed</th>
<th>Passed</th>
<th>New</th>
</tr>
</thead>
<tbody>
<tr>
<td>23-Apr</td>
<td>52</td>
<td>16</td>
<td>-</td>
<td>-</td>
<td>36</td>
<td>52</td>
</tr>
<tr>
<td>26-Apr</td>
<td>87</td>
<td>24</td>
<td>8</td>
<td>16</td>
<td>63</td>
<td>35</td>
</tr>
<tr>
<td>27-Apr</td>
<td>88</td>
<td>24</td>
<td>0</td>
<td>24</td>
<td>64</td>
<td>1</td>
</tr>
<tr>
<td>6-May</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>13-May</td>
<td>88</td>
<td>24</td>
<td>0</td>
<td>24</td>
<td>64</td>
<td>0</td>
</tr>
<tr>
<td>17-May</td>
<td>88</td>
<td>4</td>
<td>1</td>
<td>3</td>
<td>12</td>
<td>0</td>
</tr>
</tbody>
</table>

An examination of the test-by-test breakdown (not included here) shows that the failed tests follow no discernible pattern and merely represent bugs in the system yet to be ironed out. The major lesson from these results is that the hardware difficulties encountered by the System Software team required their full attention and thus many of the relatively simple bugs remaining in the system were not removed.

**Product State at Time of Delivery**

On the System Software side, the API and serial protocol are in full working order, as are the associated GUI and text UI. However, the RP10 platform is not functional. The current hardware issues involve three fuses that regulate power to motor modules 1, 3, and 4. It is believed that the H-Bridge's for these motor modules are causing a short circuit, by telling the motor to spin forwards and backwards at the same time. However, nothing is confirmed. Our team had limited electrical knowledge and not enough time to properly diagnose the problem.

Unplanned features added include Lego MINDSTORMS support to the API and GUI as well as a new protocol designed for the MINDSTORMS robot – all added during the final week to facilitate a demonstration during the final presentation. A user manual is available for the API. The test suite is also available in the System Software package.

The Model deliverables are partially complete. The SolidWorks model is complete and ready to be inserted into MRDS. The platform characteristics document, while untested (again due to the unavailable RP10), is also complete. The simulation is incomplete, however. The services and entities, while correct to the best of the team’s knowledge, are not fully functional. The services run, the entities appear onscreen (in the form of the imported SolidWorks model), but the VPL application has no visible driving effect on the simulation. Much of the failure to deliver is described above in the schedule discussion. For the System Software team, incomplete deliverables are explainable due to the persistent failures of the hardware. For the Model team,
the incompletion of the desired simulation lay in the difficulties of grasping the full extent of the MRDS CCR and DSS architectures and of applying them to the current problem. The platform characteristics could also not be tested due to the nonfunctional RP10.

To pass knowledge to a future team, supplemental documents were created. For example, Kyle wrote instructions for importing a model from SolidWorks to MRDS through Blender, an open-source 3-D modeling application. The goal is to provide more useful documentation that can better prepare the next team in early project planning and learning.

Project Reflection

By and large, the team is satisfied with the application of the Spiral model in the second half of the project. By focusing on risks and actively tracking tasks, these process activities provided a solid framework with which to judge progress both individually and as a group. The API and protocol designs also functioned well, as evidenced by the rapid adaptation to Mindstorms in such a short time. The team also was pleased with the integration of the Mechanical Engineering students in the team dynamic.

Unfortunately, this project tested our patience time after time. Some of the project’s disruptions include excessive hardware difficulties, MRDS’s unexpected lack of 3-D model design functionality, MRDS’s poorly documented service/entity/proxy architecture, and the lack of support from all sources in getting SolidWorks 2009.

As for next time, let us suggest the following:

1. Delays in schedule for System Software tasks stemmed from not having someone qualified to diagnose and/or repair the RP10 when electrical issues were realized. The team understands that having one or two electrical or computer engineers on the team to “babysit” the robot is unrealistic (and, in fact, something of an insult to the engineers in question). Perhaps two or three software engineers should be involved in a KGCOE Multidisciplinary Senior Design team. The software stack would be developed alongside the construction of an improved robot. A functional platform would not be available for most of the two project quarters, but hardware issues could be resolved quicker. The software engineers would have to prototype using an alternative working platform like Lego Mindstorms.

2. Plan for risks that seem only remotely possible (as opposed to ridiculous like the threat of nuclear annihilation). It is instinctive to say that issues will be resolved over time or major problems will simply not occur. It is easy to disregard risks that are classified as low priority. However, the team should seriously consider recovery strategies. By thinking what must be done after a problem has occurred, project plans could change significantly. Recovery strategies might be more important than mitigation strategies that build a team mentality that problems will most likely be prevented. Also, seriously adhere to a deadline for resolving risks to try recovery strategies.