ECE 492

Directed Research Study

Microsoft Robotics Studio

Final Report
By Hussain Al-Helal
Abstract

The following paper evaluates the strength of the Microsoft Robotics Studio (MRS). Comparisons between MRS and existing robotics projects will be made based on the simplicity and robustness of the software construction, simulation and interfacing with hardware. A number of example software exercises will be highlighted in order to draw an accurate comparison between MRS and alternate robotics projects. The design and simulation of an autonomous robot is documented within the paper. MRS is found to be both simple enough for first time robotics developers and robust enough for the robotics enthusiast.

Introduction

With the advent of robotics, several projects have been developed. For those with experience of the Orca robotics project it can be fascinating to develop robotics software with something different. Microsoft Robotics Studio (MRS) is a robotics project aimed at allowing a wide variety of users the ability to design robotics applications. User’s from academic researchers and robotics experts to hobbyists and first time robotics users are Microsoft’s target audience.

MRS is available as a free download. Multiple applications are included with the install, a Visual Programming (VPL) environment as well as a simulation environment that is driven by AGEIA’s physics engine giving realistic simulation environments. It is possible to run MRS on multiple operating systems including “Windows CE; Windows Server 2003 R2 (32-Bit x86); Windows Server 2003 R2 x64 editions; Windows Vista; Windows XP; Windows XP 64-bit”1.

The VPL environment is probably the most interesting aspect of MRS; it sets this project apart from other available robotics projects. VPL is a “graphical data-flow-based programming model”2 giving a more intuitive programming environment for first time programmers. Being able to visualise a program as a set of connected data-flow diagrams vastly decreases the learning curve of a new programming language. This may prove to make MRS a very powerful project.

Creating Robotics Applications with MRS

MRS can be linked with Microsoft Visual Studio in order to produce robotics applications using any of the supported programming languages in Visual Studio. As many existing robotics projects allow you to program in these languages it is more interesting to create robotics applications using the built in VPL environment as this truly sets MRS apart from its alternatives. In order to develop a robotics application using MRS it is required to open the VPL environment. The following example will detail how to design and simulate a robotics application to explore an unknown environment and avoid collisions through the use of a laser range finder sensor.

![Microsoft Visual Programming Language Environment](image1.png)

**Figure 1 - Microsoft Visual Programming Language Environment**

Once inside the VPL environment it is possible to recognise two important contributors to the application: Basic Activities and Services. In order to design an application using the environment, one must simply orchestrate the input and output of such activities and services. In order to accomplish the intelligent use of a laser range finder, a definition for the activity must be defined. This can be achieved by simply dragging the “Activity” basic activity into the diagram.

![Description of Activity](image2.png)

**Figure 2 - Description of Activity**
Once the activity is in the diagram the ability to define the behaviour becomes evident.
By highlighting the activity it is possible to set a name and description. The user can then
double click the activity to define its behaviour.

![Image of activity description]

**Figure 3 - Describing an activity**

Double clicking on the activity opens a new diagram. The functionality of the activity is
defined within this diagram. The activity can then be reused anywhere within the project
whilst maintaining the defined behaviour.

![Image of new diagram]

**Figure 4 - New diagram for defining the behaviour of an activity**

Assuming the laser rays are partitioned into three sections; left, right and centre it is
desirable to detect objects and determine the relative position of the object with respect to
the laser. In order to do this, the following algorithm has been devised.

```plaintext
LaserRangeFinder

While !FoundNearest
  ray++;
  if ray.distance > limit
    return ray.side;
  end While;
else
  if ray.distance < Nearest and ray.distance > 0
    ray = ray;
  else
    ray = Nearest;
end if

Algorithm 1 - Laser Range Finder
```

University of Arizona 4 ECE Department
The idea of Algorithm 1 is to scan each laser ray until an object has been detected with a distance closer than the predefined limit and then return the partition of the laser rays it was read from. An example would be, if a robot was maneuvering and an obstacle was present on the right of the robot, the laser would detect the obstacle is on the right and the necessary maneuver can be carried out to avoid the obstacle. It is actually extremely simple to orchestrate the algorithm using VPL. Based on the definition connections on the left signify inputs whereas connections on the right signify outputs it is possible to implement the algorithm as follows:

Initially the inputs and outputs for the diagram should be defined. They can be defined by selecting the button next to Action in the upper left of the diagram and then filling in the details. By selecting add it is possible to add actions to the diagram as well as inputs and outputs as shown in Figure 6. By selecting the Notifications tab you can also define any signals to be sent. In this example we will send a signal FoundNearest when the nearest object is found to be closer than it should be.

Select the left most connection of the LaserRangeFinder diagram as shown in Figure 7.
After dragging the connection and placing it in the diagram you are prompted to select an activity from the list.

For this particular connection the “If” activity is selected. Note the logic can be added to the if statement by simply typing it into the space provided by the activity. The semantics of the If activity are that the upper most output connection is used for when the logic is evaluated as true and the lower most output connection is used when the logic is evaluated as false. The output ports will serve as inputs to the corresponding activities.

By dragging the true connection of the output of the “if” statement it is possible to send messages to other activities/services outside the LaserRangeFinder activity. Select the FoundNearest notification when prompted after connecting the output to the circular output of the diagram.
Using the same principles, the rest of the diagram can be constructed as shown in Figure 12.

Figure 12 shows a fully functional LaserRangeScanner activity. Note the simplicity of orchestrating the inputs and outputs for each activity. Iterative algorithms can be implemented by calling the activity within the activity as shown in Figure 12 with LaserRangeScanner.
Once the behaviour of LaserRangeScanner has been defined it is possible to return to the main diagram tab and formulate the rest of the application.

**Algorithm 2 - Retrieving Laser Data and Determining How to React**

<table>
<thead>
<tr>
<th>Main</th>
</tr>
</thead>
<tbody>
<tr>
<td>While Driving</td>
</tr>
<tr>
<td>Retrieve Nearest Object from Laser range finder</td>
</tr>
<tr>
<td>Calculate output of laser range finder, mask as integer variable</td>
</tr>
<tr>
<td>Set masked output as input to robotic drive motor</td>
</tr>
</tbody>
</table>

**Figure 13 - Switch Activity receiving input from LaserRangeFinder**

Figure 13 shows how a switch activity can be used to retrieve data from another activity. According to Algorithm 2 it is necessary to mask the output from the laser as an integer variable so that it can be passed to the drive motor. Figure 14 illustrates how this is done.

**Figure 14 - Main diagram implementing Algorithm 2**

Figure 14 encompasses an activity labelled “AutoDrive”. This activity was constructed just as the LaserRangeFinder was earlier. The next step is to define the behaviour of
AutoDrive by double clicking on the activity; this will open a new diagram. Navigate to the diagram where the behaviour of the activity can be specified.

![Actions and Notifications](image)

**Figure 15 - Specifying Actions for the AutoDrive Activity**

Note that no outputs will be provided from this activity hence there is no reason to define a notification. The next step is to formulate an algorithm which will drive the robot according to the retrieved laser data.

<table>
<thead>
<tr>
<th>AutoDrive</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>While Receiving Laser Data</strong></td>
</tr>
<tr>
<td>Retrieve nearest obstacle distance</td>
</tr>
<tr>
<td>Determine position of obstacle</td>
</tr>
<tr>
<td>If nearest obstacle in centre AND robot is moving</td>
</tr>
<tr>
<td>If obstacle is not close enough</td>
</tr>
<tr>
<td>Continue moving</td>
</tr>
<tr>
<td>Else</td>
</tr>
<tr>
<td>Stop robot</td>
</tr>
<tr>
<td>Else</td>
</tr>
<tr>
<td>Determine whether obstacle is closer to left or right</td>
</tr>
<tr>
<td>Set nearest obstacle distance as determined</td>
</tr>
<tr>
<td>If nearest obstacle on left</td>
</tr>
<tr>
<td>Turn right until nearest obstacle in centre</td>
</tr>
<tr>
<td>Else</td>
</tr>
<tr>
<td>Turn left until nearest obstacle in centre</td>
</tr>
</tbody>
</table>

**Algorithm 3 – Avoid Collisions Based on Laser Data**

Algorithm 3 can be realized using VPL as shown in Figure 16.
Notice how in Figure 16 a GenericDifferentialDrive service has been placed. Services relate to physical hardware in most cases and must be associated with a manifest if they are to be simulated or created in hardware. The manifest is a simple XML file defining the device driver so that messages can be both passed and retrieved from the service at runtime. In order to associate a manifest with the service, simply select the service and on the rightmost side of the environment select “Use a manifest” and then import the appropriate manifest. The one used for this example is demoscene.Manifest.xml.

Now that the application is completed it would be beneficial to simulate the code and view the effectiveness of the algorithms implemented. Running the simulation is made
simple, by selecting the manifest chosen in Figure 17 it has been specified that the application is to run in simulation using a Sick LMS laser range finder mounted on a Pioneer 3DX robot.

By clicking on the green “Run” button the code is compiled and the simulation is started. Note that any errors will appear in the DSS Runtime prompt highlighted in red and sections that pass will be highlighted in green.

![Figure 18 - DSS Runtime prompt shown when application is run](image-url)
Figure 19 and 20 display the main camera which can be placed within the manifest. This particular manifest allows for a second camera. One mounted on the robot itself. It is extremely simple to view either camera by selecting the Camera tab and selecting your desired camera view from the drop down list.
Figure 21 - Selecting camera view

Figure 22 - Simulation Environment with RoboCam 1

Figure 23 - Simulation Environment with RoboCam 2
Editing Robotics Applications with MRS

Once you have a working application it is simple to go back and edit the program. The data-flow diagrams make it simple for users to follow the semantics of the application and make the necessary adjustments. With the example highlighted above it is possible to add a simple dashboard to the simulation so the laser data can be used to visualise the surroundings.

By navigating to the diagram tab it is a simple measure of adding the Simple Dashboard service and then setting the manifest as SimpleDashboard.xml.

![Diagram](image-url)

Figure 24 - Adding Simple Dashboard to the example
Figure 25 - Associating SimpleDashboard.Manifest.xml with Simple Dashboard service

Running the application now opens a dashboard as well as the simulation environment. It is necessary to connect the dashboard to the robot so the laser data from the simulation can be ported to the dashboard. This is done by entering the machine and port data.

Figure 26 - Simple Dashboard Interface
Figure 27 - Simple Dashboard capturing Laser Range Finder Data

Note that in Figure 27 the machine name is entered and port 50001 is selected. This allows the dashboard to communicate with the application making it possible to view the laser data.

Comparison to Existing Robotics Projects

MRS is only one of several robotics projects; Orca is another powerful project being used by many robotics developers. The Orca project is open source and allows users to edit/contribute components to suit their individual needs. The majority of drivers are written in C++ and it is possible to develop the application using Linux. The project relies heavily on the users abilities in Linux and their competence using Linux Terminal. A noticeable difference is evident from the start; C++ has quite a large learning curve for users with little/no programming experience. Inexperienced users can refer to documentation available from the Orca robotics website but the tutorials are limited and finding the documentation can be non-intuitive as most of the useful documentation is not available through Orca. For experienced programmers, Orca allows for development of highly powerful robotics applications such as autonomous vehicles displayed in the
“Darpa Grand Challenge”\(^3\). Software development can be performed much faster in MRS for simple applications such as drive-by-wire.

It is possible to interface the Orca robotics project with several other open source projects including Gazebo. The Gazebo project allows users to simulate their robotics applications in 3D similar to the environment provided in MRS. Installing several different open source projects can be time consuming and interfacing the projects requires a lot of reading in order to execute the simulation efficiently. This may seem like a lot of unnecessary work to those with prior experience with MRS. The ability to run orca in Linux has several benefits. It is possible to port the application onto an embedded Linux machine such as a Gumstix and autonomously control a robot. This provides a low cost solution for wirelessly communicating with the robot. MRS cannot be ported to Linux but it can be ported to an embedded Microsoft device to similar effect.

By harnessing the VPL contained in MRS it following code has become much simpler. Orcas’ open source approach means that many developers with very different programming styles will be downloading and developing many existing robotics components. The documentation and style of the programmer writing the components heavily influences the ability for others to develop the code further. Large C++ files and other text based programming languages can be time consuming to understand and follow as is the case with the Orca project. MRS’ data-flow diagrams make it much less time consuming understanding the structure of the program.

By being able to follow the flow of the programs much easier using VPL it is also much simpler to edit existing applications. Searching through dense code for every instance of an attribute has become non-existent with VPL.

On the other hand, MRS is still at an early stage in its release and not all services that come supplied with the software work “out of the box”. An example of this is the GPS service that is offered. The service currently only supports hardware implementations and cannot be simulated without major programming in a text based language, negating the simplicity factor of VPL.

The potential for the Orca project is phenomenal but as a robotics project on its own, it is relatively limited. In order to simulate components meaningfully from the Orca project they must be interfaced with the Gazebo project, if the components have to be synthesised onto hardware then they must be interfaced with the Hydro project. For anything to be seriously effective from the Orca project the developer must have

```c++
namespace laser3dstatic {

Driver::Config::Config()
    : minRange(0.0),
     maxRange(0.0),
     detectorPitch(0.0),
     focalLength(0.0),
     numberOfColumns(0),
     numberOfRows(0),
     numberOfSamples(0)
{}

bool Driver::Config::validateData() const
{
    if (minRange <= 0.0) return false;
    if (maxRange <= 0.0) return false;
    if (detectorPitch <= 0.0) return false;
    if (focalLength <= 0.0) return false;
    if (numberOfColumns <= 0) return false;
    if (numberOfRows <= 0) return false;
    if (numberOfSamples <= 0) return false;
    return true;
}

std::string
Driver::Config::toString() const
{
    std::stringstream ss;
    ss << "Laser driver config: min_range=" << minRange << " max_range=" << maxRange << " detector_pitch=" << detectorPitch << " focal_length=" << focalLength << " number_of_columns=" << numberOfColumns << " number_of_rows=" << numberOfRows << " number_of_samples=" << numberOfSamples;
    return ss.str();
}

bool
Driver::Config::operator== (const Driver::Config &other)
{
    if (other == this)
    {
        return true;
    }
```

Figure 29 - Sample Code from laser3dstatic component driver
knowledge of not only the Orca project but also competency with the Hydro and Gazebo project. MRS integrates the code development, simulation and integration onto hardware in one development studio making it much simpler to manage all three tasks.

![Gazebo Simulator](image)

**Figure 30 - Gazebo Simulator**

**Conclusion**

It seems like MRS is a much more efficient environment for rapidly developing robotics prototypes; translating a design into a data-flow diagram is much more intuitive than translating it into C++ or other low level programming languages.

By introducing the VPL environment MRS is much more appealing to beginner programmers; the learning curve is much smaller than that of Orca. MRS also allows robotics applications to be developed in any of the Microsoft programming languages. This allows for robotics applications to be developed in a similar way to Orca but with the added choice of using VPL.

The simulation environment in MRS is much more powerful than that of Gazebo. The use of the AGEIA physics engine makes this one of the most powerful simulation environments in any of the robotics projects due to inherent hardware acceleration and accurate physics.

Interfacing software and hardware is very simple using MRS and can be done in seconds. Orca takes much more time to interface software and hardware and is not as easy for a typical robotics hobbyist; a large amount of robots are already supported in MRS and several example manifests are available and easily updated to support newer robotics applications.
Both projects are very powerful robotics projects. MRS is something that can be learned much faster than Orca. MRS can also be used to develop prototypes for robotics much faster than Orca. The introduction of VPL makes MRS a very powerful tool for robotics development. Although VPL is not yet as robust as languages used in Orca MRS allows for robotics projects to be developed in all Microsoft programming languages. This allows projects developed in MRS to be just as robust as those developed in Orca. The Orca project is very user friendly for Linux super users who understand the file based system and can traverse the system using the Linux Terminal competently. MRS is more suited to users that are comfortable with graphical representations such as flow diagrams and those who have prior experience with software such as Simulink. The real tradeoff between the two is whether the user wants to rapidly create an application. If this is the case then the user should choose MRS as their development environment. If the user is confident with Linux and has a deep knowledge of low level programming languages as well as more time to spend on the application then they may choose to use the Orca development environment.