Introduction to
IoT and RaaS Programming with ASU-VPL

http://venus.eas.asu.edu/WSRepository/ASU-VPL/

Yinong Chen and Gennaro De Luca

School of Computing, Informatics, and Decision Systems Engineering
Arizona State University

Microsoft VPL is a milestone in software engineering and robotics from many aspects. It is service-oriented; it is workflow-based; it is event-driven; it supports parallel computing; and it is a great educational tool that is simple to learn and yet powerful and expressive.

Sponsored by two Innovation Excellence awards from Microsoft Research in 2003 and in 2005, Dr. Yinong Chen participated in the earlier discussion of service-oriented robotics at Microsoft. Microsoft VPL was immediately adopted by Chen in developing the freshman course CSE101 in 2006. The course grew from 70 students in 2006 to over 350 students in 2011. The course was extended to all students in Ira A. Fulton Schools of Engineering at ASU and was renamed FSE100, which is offered to thousands of freshman engineering students now.

Unfortunately, Microsoft stopped developing and supporting VPL recently, which lead to our FSE100 course, and many other schools’ courses using VPL, without further support. Particularly, the current version of VPL does not support LEGO’s third generation of EV3 robot, while the second generation NXT is out of the market.

To keep our course running and also help the other schools, we take the challenge and responsibility to develop our own visual programming environment, ASU-VPL, based on the Robot as a Service concept. The purpose of this project is to provide a free environment supporting the VPL development community in education and research. To serve this purpose, ASU-VPL keeps the great features that VPL has and provides a similar user interface and functionality, so that the MRDS and VPL development community can use ASU-VPL with no learning curve. ASU-VPL does not replace Microsoft VPL. Instead, it extends VPL in its capacity to connect to different physical robots, including EV3 and Intel-based open architecture robots.

Robot as a Service and the ASU-VPL environment are designed and developed by Yinong Chen and Gennaro De Luca, with contributions from Calvin Cheng, Megan Plachecki, and Sami Mian. ASU-VPL
documents, software, and sample code can be downloaded from the ASU Web Service and Application repository. The direct link is http://venus.eas.asu.edu/WSRepository/ASU-VPL/

1. **ASU-VPL versus Microsoft VPL**

ASU-VPL uses the same computing model as Microsoft VPL. The program is running on a Windows computer, a desktop, a laptop, or a tablet. The computer sends commands to control the robot actuators (motors) and receives the sensory data and motor feedback from the robot. The data between the computer and the robot is encoded in a JSON object which is in plain text format. It supports Wi-Fi, Bluetooth and USB connections between the main computer and the robot. ASU-VPL supports EV3 and any self-developed robots. We have developed different robots based on Intel architecture, the Linux operating system, and the Windows operating system.

Figure 1 compares the activities and services between ASU-VPL and Microsoft VPL. ASU-VPL implemented most basic activities in VPL and implemented additional While, Break, and End While activities to facilitate loop building, which can reduce the circular paths in VPL diagrams.

In the current version, three sets of services are implemented. The first set is a list of generic services, including the Simple Dialog, Key Press Event (Direction Dialog), Text To Speech, and Timer. The second set is for EV3 robots. The third set of services are used to connect to generic robots, various sensors, and motor services. The significant addition to Microsoft VPL is the inclusion of LEGO EV3 services and generic robot services. In Microsoft VPL, DSS services developed specifically for MRDS can be added into the VPL service list. In ASU-VPL, SOAP and RESTful services can be added into the service list.

![Figure 1. Activities and services: ASU-VPL versus Microsoft VPL](image-url)
Many improvements are made in ASU-VPL. For example, ASU-VPL use state.varaible consistently. In VPL, local variables (the location of a variable’s use has a direct link to the variable) use variable name only, while and global variables use state.varaible syntax. It is not easy for inexperienced programmer to recognize the scope of variables and often causes confusion.

2. Getting Started with ASU-VPL

Download ASU-VPL software, sample codes, and tutorial at:

http://venus.eas.asu.edu/WSRepository/ASU-VPL/

Unzip and open the folder Release, start the ASU-VPL software from the file: VisualProgrammingEnvironment. Now, you can learn ASU-VPL by following the tutorial exercises.

ASU-VPL is general purpose programming language and is turning-complete in its capacity. It can be used for any kind of computation tasks. In ASU-VPL, A program is represented as a diagram of workflow, and the components are defined as activities in the diagram. We will learn the language through a series of examples and exercises.

Exercise 2.1 Hello World in ASU-VPL

ASU-VPL is similar to Microsoft VPL not only in concepts but also in programming. It is our intention to have Microsoft VPL programmers use ASU-VPL with little learning. We now show examples of basic programming in ASU-VPL. We start with the Hello World program. Figure 2 shows the two versions of code using VPL and ASU-VPL. The two diagrams look the same. However, ASU-VPL has simplified a couple of steps: it automatically changes the type to String after a string is entered, and the default null value step in Microsoft VPL is eliminated.

![Figure 2. Hello world program in (a) Microsoft VPL and (b) ASU-VPL](image)

Exercise 2.2 Input and Output in ASU-VPL

Now, we use ASU-VPL to implement the examples that we have implemented Microsoft VPL in the previous sections. Simple Dialog service in ASU-VPL can be used for both input and output. The diagram in Figure 3 uses Simple Dialog for input first. The connection type is PromptText. The following three dialog boxes in Figure 3 show examples of the data connection, input prompt when executing the code, and the output. The same diagram also uses Simple Dialog for output when the connection type is set to AlertDialog.
Exercise 2.3 A Simple Counter in ASU VPL

Now, we can implement a simple counter in ASU-VPL diagram, as shown in Figure 4. A similar implementation of the counter activity in Microsoft VPL can be found in our VPL tutorial.

Exercise 2.4 Using While-Loop to implement Counter

We formed a loop by ourselves in the previous exercise. We can use a built-in loop in ASU-VPL to make the program structure simpler. The diagram is shown in Figure 5.
Figure 5. Using while-loop for implementing the counter

Quiz: What is the main different between this counter and the counter in the previous exercise?

Exercise 2.5 Counter as an Activity in ASU VPL

As a more complex example, Figure 6 shows the ASU-VPL program that implements: (a) a main diagram and (b) a counter activity. The activity takes an input N in the main diagram. In the activity CountToN, it starts from 0 and adds 1 in each iteration. It stops when the counter value is equal to N. Text to Speech service is used to read out the numbers in the activity, and Simple Dialog Service is used to print the counted numbers. As can be seen in the activity diagram, both data output and notification (event) are supported in ASU-VPL. A similar implementation of the counter activity in Microsoft VPL.

Quiz: What is the main different between this counter and the counter in the previous two exercises?

Although IS-VPL can be used as general programming language, its strength is in event-driven programming that can response to a sequence of events. The event-driven applications are best described by finite state machines.
consisting of states and transitions between the states. The transitions are triggered by events. We will start to use ASU-VPL for solving event-driven problems.

**Exercise 2.6 Implement a Vending Machine**

Given a Finite State Machine (FSM) in the following diagram, implement the FSM-defined vending machine using ASU-VPL.

![Finite State Machine of a vending machine](image)

A sample diagram is shown as follows.
Exercise 2.7 Garage Door Opener

Given a Finite State Machine (FSM) in the following diagram, implement a simulated garage door control logic in ASU-VPL.

The remote controller is a touch sensor or a key press event, and the limit sensor is a build-in sensor in the motor. When the door stop, the limit sensor will generate a notification. You can start your program similar to the maze navigation program we learned in Microsoft VPL, as shown in the diagram below.
The limit sensor built-in the motor can be simulated as follows. When key m is pressed while the door is opening or is closing, the state will change and result will be displayed.

At this time, ASU-VPL does not have the simulated sensors and simulated drive (motors), we have to use key press event and Print Line to simulate the sensors and motors.

Exercise 2.8 Parity Detection

Given a Finite State Machine (FSM) in the following diagram, implement a simulated parity detection logic in ASU-VPL. The program must generate a 1 output if the number of 1s entered is an even number, otherwise, it must generate a 0 output.

One of the purposes of ASU-VPL is to extend Microsoft VPL to more conveniently support EV3 and other physical robots. In the rest of the section, we will focus on using ASU-VPL to connect to different physical robots. We start with the drive-by-wire program that controls the EV3 robot using the keyboard of the computer.

3. Controlling EV3 Robots Using ASU-VPL

In this section, we start to use ASU-VPL to write programs to control EV3 robots.
Exercise 3.1 Build Your EV3 Robot

If you have an EV3 robot, you can build your robot use the book in the EV3 box, or follow the link: http://robots2doss.org/?p=133

Exercise 3.2 Drive-EV3-by-wire in ASU VPL

Figure 3.1 shows the program that remotely controls an EV3 robot using the four arrow keys on the computer’s keyboard. The Key Press Events are services that offer the same functionality as that of the Direction Dialog in Microsoft VPL. As we can define the key individually and we can define more than four keys, using Key Press Events is more flexible than using Direction Dialog.

Quiz 1: How do you make the robot to turn left? How do you make the robot to turn right?

Quiz 2: What could be a problem in driving the robot using this program?

Figure 3.1. Drive an EV3 robot by wire
Exercise 3-3 Connecting robot to Computer via Bluetooth or Wi-Fi

To make the diagram work with a physical robot, we need to configure the devices used in the diagram in a number of steps.

**Step 1:** Configure the EV3 brick. Similar to the Microsoft VPL code, we use a main brick, called My EV3 Brick, to define the major configuration. It is possible to add more than one brick into a diagram. Right-click the brick to open the configuration. Figure 8 shows the right-click window and the two configuration windows. Change Connection Type window allows us to choose one of three available connection methods: Wi-Fi, Bluetooth, and USB. If Bluetooth is selected, the standard Bluetooth pairing process can be used to establish the connection between the computer running the ASU-VPL program and the robot.

![Configuration of the EV3 brick using Bluetooth connection](image)

**Figure 3.2.** Configuration of the EV3 brick using Bluetooth connection

**Step 2:** Configure the other devices. For each device, we need to choose the partner and choose the connection ports that the devices use. There is one device used in this diagram: the Drive service. Figure 9 shows the right-click window to the EV3 Drive service. The configuration sets the Drive service to partner with My EV3 Brick 0, and we assume that the Drive wheels are connected to the motor ports B and C on the EV3 Brick 0, respectively.

![Configuration of the EV3 brick](image)

**Figure 3.3.** Configuration of the EV3 brick
**Step 3:** Establish the connection. We assume that Bluetooth connection is selected. Bluetooth connection process depends on the Bluetooth device installed on your computer. A typical process consists of the following steps:

1. Open the Bluetooth panel from the computer’s Task Bar, and choose Add a Device.
2. The Bluetooth panel will show the devices that are ready to add. Find the EV3 robot that you want to add. Note, you can see the robot name on your EV3. You can also change name to make it unique.
3. Once the add request is issued from the computer, the EV3 side will pop up a confirmation check box. Confirm it. EV3 will further generate a pass code 1234. Confirm the code again.
4. On the computer side, a textbox will then pop up. Enter the pass code 1234 to connect.
5. After the connection, we still need to know the “Outgoing” COM port of the connection. This port needs to be entered into the Property window in My EV3 Brick 0, as shown in Figure 6. To find the COM port, open the Bluetooth panel from the Task Bar and choose Open Bluetooth Settings. We will see the COM Ports, as shown in Figure 3.4.

![Figure 3.4](image)

*Figure 3.4. Find the COM port for My EV3 Brick 0 configuration*

Once the configuration is completed, we can start to run the program and use the keyboard to drive the robot forward, backward, left, right, and stop.

We do not use Wi-Fi here. However, if Wi-Fi is selected for connection, as shown in Figure 3.5, we need to find the IP address of the robot and enter it into the Properties window so that the ASU-VPL program can establish the Wi-Fi connection to the robot. Using the buttons and the screen on EV3, you can choose the Wi-Fi network to connect to and find the IP address after the connection.
Exercise 3-4 Improving the Driving Experience

It is hard to control the robot using the program in the previous exercise. The main problem is that the robot does not stop or slow down when the key is released. We can improve it by removing the drive power when the key is released, as shown in Figure 3.6.

What does the Print Line service do in this program?
Exercise 3-5 Finding sensor reading from EV3 Brick

For color sensor and the ultrasonic sensor, sensor reading values (light reflection from the floor and the distance to the obstacle) can be found from EV3 brick. You can use the EV3 button and screen to do the calibration. As shown in Figure 3.7, first, have the color sensor face the floor, use the left-button to select the third tag, and then, select “Port View”. It will show the sensor reading value. Then, move the color sensor head to the black line, and repeat the process to read reflection value. Move to the next port, you will see the ultrasonic sensor reading value.

![Figure 3.7. View sensor reading values from EV3 Brick screen](image)

Exercise 3-6 Finding sensor reading by a program

You can write an ASU-VPL program to find the sensory value. Figure 3.8 shows the code for EV3 sensors. A similar program can be written for Edison robots.

![Figure 3.8. View sensor reading values using s program](image)

Exercise 3-7 Testing more sensors

For EV3 or Edison robot, adding more sensors into the program and test them.
Quiz: What values you receive from the tests?

Exercise 3-8 Line-follower in ASU-VPL

Using a color sensor facing to the floor, the robot can recognize the line and move following the line on the floor. The color sensor reads the light reflection from the floor. Depending on the color of the floor and the color of the line, you need to find the reflection values. You can use the buttons and screen on EV3 brick to do the sensor value calibration.

Figure 3.9 shows ASU-VPL code that makes the EV3 follow a black line on the light-brown floor. The initial position of the color sensor is on the black line or on the right side of black line. The If-activity checks the sensor reading. If the reflection value is less than 20 (reading from black line), and the robot is not in Adjusting state, the robot will turn right for 200 milliseconds and then move straightforward. This action will make the robot moving away from the black line, resulting the sensor reading value to be greater than 20. In this case, the robot will turn left and move towards the black line. When the sensor sees the black line, it starts to move away from the black line. The variable Adjusting is used to make sure that the robot completes the 200 milliseconds adjustment before it turns right.

The program does not have a loop. It is event-driven. The event source is the color sensor, which generates events periodically. Whenever an event happens, the If-activity will be triggered and executed.

![Main Diagram]

Figure 3.9. The line follower program in ASU-VPL

Figure 3.10 shows the color sensor properties and the three EV3 Drive power values that control the robot turning right, moving straightforward, and turning left.
Figure 3.10. Sensor properties and the Drive power settings

Exercise 3-9 Wall-following maze navigation in ASU-VPL (main diagram)

First, we define the right-wall-following algorithm using the finite state machine in Figure 3.11.

Two variables are used. An integer type variable BaseDistance stores the desired distance to the wall, which is initialized to 50 millimeter or the initial measurement of the range sensor. A string variable Status is used to store the moving status of the robot, and it is initialized to “Forward.” The Status variable can take one of these values: “Forward,” “TurningLeft,” “TurnedLeft,” “TurningRight,” and “TurnedRight.” We do not need a status value for turning “Left 1 degree” or “Right 1 degree,” as these two actions can be done instantly, and other actions do not need to coordinate with these two status. The execution process can be described in the following algorithm.

1. Variable BaseDistance = 400 (or initially measured distance);
2. The robot repeats all the following steps in a loop, until the touch-sensor-pressed event occurs;
   2.1) Status = “Forward”; Robot moves forward;
   2.2) Robot keeps measures the right side-distance in certain interval defined by polling frequency, and it compares the newly measured distance with the distance stored in variable BaseDistance;
   2.3) If the distance measured is less than BaseDistance - 5, the robot turns one degree left, and then returns to step 2;
   2.4) If the distance measured is greater than BaseDistance + 5, turns one degree right, and then returns to step 2;

Figure 3.11. Finite state machine describing the right-wall-following algorithm
6) If the distance measured greater than BaseDistance + 200, Status = “TurningRight90”; start to turn 90 degree right; after turning, Status = “TurnedRight”; and then returns to step 2;

3. Touch sensor is pressed; robot moves backward 0.5 rotations, Status = “TurningLeft90”; start to turn 90 degree left; after turning, Status = “TurnedLeft”; and then returns to step 2;

Following the finite state machine, we now can write the code for a robot to autonomously navigate through a maze. Figure 3.12 shows the main diagram that implements the finite state machine. A main brick, an ultrasonic sensor, a touch sensor, and a drive service are used in the program.

**Figure 3.12.** The main diagram of the right wall-following program

**Exercise 3-10 Write the Init Activity**

**Figure 3.13.** The Init Activity
Exercise 3-11 Write the Left1 Activity

The data connection for the two driver services are as follows.

![Diagram of the Left1 Activity and Data Connection]

Figure 3.15. Figure 3.14. The Left1 Activity and Data Connection

Exercise 3-12 Write the Right90 Activity

The data connection for the two driver services are as follows.

![Diagram of the Right90 Activity and Data Connection]

Figure 3.15. Right90 Activity and Data Connection
Exercise 3-13 Write the Left90 Activity

Based on the code Right90 to implement the Left90

Exercise 3-14 Write the Right1 Activity

Based on the code Left1 to implement the Right1

Exercise 3-15 Write the Backward Activity

Use negative power for both wheels

![Diagram of the Left90 Activity](image)

**Figure 3.16.** Backwards Activity

Exercise 3-16 Write the ResetState Activity

![Diagram of the ResetState Activity](image)

**Figure 3.17.** ResetState Activity
Exercise 3-17 Write the Forward Activity

**Figure 3.18. Forward Activity and Data Connection**

Exercise 3-18 Configure Sensors in Wall-following maze navigation in ASU-VPL

Similar to Microsoft VPL, activities that define components can be defined in ASU-VPL. In the previous exercises, the codes of the activities, Init, Backward, and Left1 are developed.

For each sensor, we need to choose the partner and choose the sensor port that the device uses. For the main brick and the drive service, the same configurations are used as used in Figure 6, Figure 7, and Figure 8. Figure 3.19 shows the right-click window to configure the EV3 Ultrasonic distance sensor and the Touch sensor, respectively. The configurations assume that all the devices will partner with My EV3 Brick 0, the Ultrasonic sensor is connected to sensor port 3, and the Touch sensor is connected to sensor port 4 on the EV3 Brick.

**Figure 3.19. Configurations of the EV3 Ultrasonic sensor and Touch sensor**

The full code of right wall-following program and the video of the robot’s navigating the maze, as well as other sample code, can be found in the ASU-VPL site: [http://venus.eas.asu.edu/WSRepository/ASU-VPL](http://venus.eas.asu.edu/WSRepository/ASU-VPL).
Exercise 3-19 Event-driven Wall-following

Thus far, our algorithm has been implemented using only sequential programming, other than the predefined robot sensor events. Defining our own set of events provides several advantages, including clearer code. In addition, using events allows us to more closely represent the original finite state machine. Some of the transitions in the finite state machine are actually events, rather than user input.

For example, after “Turning Left 90,” we are supposed to move to “Turned Left 90” and “Forward” after the “leftFinished” event triggers. Figure 3.20 demonstrates the changes required to define this event. Inside the “Left90” activity, we will draw a line to the circle instead of the triangle. The circle represents an event, and declares that we want to trigger an event when we are done turning left 90. Since we are not drawing a line to the triangle, the activity block will never output a value to an activity connected sequentially after the “Left90” activity. Similar changes were made in the “Left1,” “Right90,” “Right1,” and “Backward” activities.

Figure 3.20. Adding an event trigger in the “Left90” activity

Now that we have defined our events, we need to handle the events. An event handler is a piece of code which is executed whenever a certain event occurs. We can define a custom event handler for our events using the “Custom Event” activity. By selecting one of our activity blocks in the drop down menu, we can handle any events triggered by that activity block.

For example, if we select “Left90” in the “Custom Event” activity’s drop down box, the code following that “Custom Event” activity will execute immediately after the code shown in Figure 3.20 reaches the circle pin. Figure 3.21 shows the updated program which uses events to transition between states instead of sequential programming. We also used a “Merge” activity to avoid repeating the “ResetState” activity.
Figure 3.21. The right wall-following algorithm with custom events

Exercise 3-20 Implementing the two distance algorithm in ASU-VPL

We use an EV3 robot in the maze navigation algorithm defined in Figure 3.22. We will use ASU-VPL to implement this heuristic algorithm. As ASU-VPL is an event-driven language, it is the best way to specify the algorithm in a finite state machine. In the diagram, we use two variables. The variable “Status” can take six possible string-type values of Forward, TurningRight, TurnedRight, TurningLeft (Spin180), TurnedLeft, and Resume180. The int-type variable “RightDistance” is used to store the distance to the obstacle after the robot turned right.

The algorithm is said to be heuristic, because it cannot find a solution in all cases of mazes. However, it has a good chance to find a solution in most mazes, given the information available collected by a single range sensor.
4. Controlling Edison Robots Using ASU-VPL

In this section, we will write ASU-VPL program to control self-built open architecture robots. We have built a robot using Intel Edison board with Linux operating system.

Exercise 4.1 Build Your Edison Robot

To assemble an Edison robot, you can start with the instructions to build an Edison board with Arduino extension board:


For constructing the robot part, you can follow the instructions given by your by your instructor. The following link shows one of the Edison robot built at ASU for the Intel Roadshow 2015:


The software installed on the Edison robot is listed as follows:

- ArduinoFiles
- node_modules
- main
- run.sh

where, the file run.sh is the script to start the program on the robot. We need to start this file before ASU-VPL can communicate the robot. The file main.js is a JavaScript program that performs most of the work on the robot. It establishes the connection with the WiFi router, displays the IP address on the screen, and waits for the host computer to connect. Notice that the Edison robot and the host computer must connect to the same router. The main.js program also interprets the JSON packets from the host computer and sends the commands to the motors. The folder node_modules contains the library functions that support the scripts in main.js. The folder ArduinoFiles contains the code that read the sonic sensor to obtain the distance value measured by the sensor.

ASU-VPL supports an open interface to other robot platforms. Any robot that follows the same interface and can interpret the commands from ASU-VPL program can work with ASU-VPL. ASU-VPL program communicates with the robot using the following JSON object, which defines the input to the robot from the ASU-VPL program and the output from the robot to the ASU-VPL program.
The ASU-VPL environment will encode the control information into this object. The robot needs to interpret the script and perform the actions defined. On the other hand, the robot will encode the feedback in the same JSON format, send that back to the ASU-VPL program, and the ASU-VPL program will extract and use the information to generate the next actions.

Sponsored by the Intel IOT Group, a number of robots based on Intel architecture, including Intel’s Galileo, Bay-Trail, and Edison, have been developed. ASU-VPL can connect to these robots via Wi-Fi or Bluetooth, send commands to them, and control them to perform different tasks.

ASU-VPL implemented two types of robots: EV3 robots and generic robots. A generic robot is a robot that can communicate with the computer running ASU-VPL and can process the JSON packet.

**Exercise 4.2 Starting the Program on Edison Robot**

The followings are the steps need to be done before your ASU-VPL program can control the Edison robot.

1. Installed Edison driver program on the computer that will communicate with the Edison board.
2. Turn on the Edison board power and motor power.
3. Wait for the robot to connect to the default router and to display the IP address.
4. Start SSH from the host computer to log into the Edison using the displayed IP address.
5. Enter the Edison’s user name, e.g., root
6. Enter the Edison’s Password, e.g., password
7. Use SSH shell command to start the program by executing ./run.sh

If the IP address does not appear on the screen of the robot, you can perform the following steps to find the IP address of the robot.

1. Connect robot to the host computer using a USB cable;
2. Start SSH from the computer to log onto the Edison using this IP Address: 192.168.2.15
   - This can only be done from a computer that has been set up according to these instructions: [https://software.intel.com/en-us/connecting-to-intel-edison-board-using-ethernet-over-usb](https://software.intel.com/en-us/connecting-to-intel-edison-board-using-ethernet-over-usb)
3. User name, e.g.: root
4. Password, e.g.: password
5. Find your IP Address using `ifconfig` command OR force it to display on the screen by running this command: `python ip.py`
6. If you need to connect to a new network, you can configure the WiFi on the Edison using this command: `configure_edison --wifi`
   - Full instructions for doing so are here: https://software.intel.com/en-us/connecting-your-intel-edison-board-using-wifi

Additional note:
- Each time you want to connect to the Edison from ASU-VPL you need to re-run the script `./run.sh`. Each script run only accepts one connection, after which it won't be able to accept another connection.

Exercise 4.3 Pairing Edison Robot with the host computer

To connect a robot to ASU-VPL program, the computer running the ASU-VPL program needs to pair with the robot. As long as a robot can (1) establish a Wi-Fi or Ethernet connection with the computer running the ASU-VPL program, (2) encode the information into the JSON object, and (3) interpret the command from the ASU-VPL program, the robot can be run from the ASU-VPL program.

Now, we can start the exercises with ASU-VPL and Edison robot.

Exercise 4.4 Drive-EdisonRobot-by-wire in ASU VPL

Figure 4.1 shows the program that remotely controls an Edison robot using the four arrow keys on the computer’s keyboard. The Key Press Events are services that offer the same functionality as that of the Direction Dialog in Microsoft VPL. As we can define the key individually and we can define more than four keys, using Key Press Events is more flexible than using Direction Dialog.

Quiz 1: How do you make the robot to turn left? How do you make the robot to turn right?

Quiz 2: What could be a problem in driving the robot using this program?
Figure 4.1. Drive an Edison robot by wire

Exercise 4.5 Intel Edison Robot in Solving the Maze Using the Two-Distance Algorithm

In this section, we will use ASU-VPL to implement the heuristic algorithm using an Edison-based robot. As ASU-VPL is an event-driven language, it is the best way to specify the algorithm in a finite state machine, as shown in Figure 4.2. In the diagram, we use two variables. The variable “Status” can take six possible string-type values: Forward, TurningRight, TurnedRight, TurningLeft (Spin180), TurnedLeft, and Resume180. The int-type variable “RightDistance” is used to store the distance to the obstacle after the robot turned right.
The finite state machines implements a heuristic algorithm that can be elaborated in the following steps.

1. The robot starts to move forward;
2. If the distance measured by the range sensor is less than 400 millimeter, it turns (90 degree) right;
3. After the event “rightFinished” occurs, it saves the distance measured to the variable RightDistance;
4. The robot then spins 180 degree left to measure the distance on the other side;
5. After the event “leftFinished” occurs, it compares the distance measured to the values saved in the variable RightDistance;
6. If the current distance is longer, it transits to the state “Forward” to move forward;
7. Otherwise, it resumes (spins 180 degree) to the other direction;
8. Then, it transits to the state “Forward” to move forward.

The algorithm is said to be heuristic, because it cannot find a solution in all cases of mazes. However, it has a good chance to find a solution in most mazes, given the information available collected by a single range sensor. For the given maze in Figure 9.36; for example, the algorithm will navigate the finish line without a problem.

Figure 4.3 shows the first part of the main diagram of ASU-VPL code.
Figure 4.3. The first part of the main diagram implementing the two-distance maze algorithm.

The algorithm starts with the robot moving forward. When it approaches a wall in the front, it measures the distance to the right and saves the distance into a variable. Then, the robot spins 180 degrees to measure the other side's distance. It compares the two distances and moves to the direction with more space. In this part of the diagram, an If activity is used to compare the current status and the distance value from the sensor, which generates four different cases.

Exercise 4.6 Controlling Intel Edison Robot in ASU-VPL

The second part of the main diagram is shown in Figure 4.4, which processes four cases of the If-activity, respectively.
Exercises 4.7 Implementing the Activities Used in the ASU-VPL Program

There are five activities are implemented to support the main diagram: Forward, Right90, Stop, Left180, and Right180. Consider each as an exercise.

Figure 4.5 shows the codes of three of these activities: Forward, Right90, and Stop. The codes of the Right180 and Left180 are similar to Right90, but with different values. A Print Line service is used for distance values for debugging purpose.
Exercises 4.8 Configure the robot

Similar to the EV3 robot, the main robot, the sensors, and the motors need to be configured. Figure 4.6 shows the configuration of the three devices.
The full code of maze navigation program and the video of the robot’s navigating the maze can be found in the ASU-VPL site: http://venus.eas.asu.edu/WSRepository/ASU-VPL/

Your Team Name:

### ASU-VPL with EV3 and Edison Robot Programming

<table>
<thead>
<tr>
<th>Date</th>
<th>Activity/Mission Completed</th>
<th>Driver Name</th>
<th>Time Stamp</th>
<th>Instructor sign</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><strong>Section 2 Getting Started with ASU-VPL</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Exercise 2-1 Hello World in ASU VPL</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Exercise 2-2 Input and Output in ASU VPL</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Exercise 2-3 A Simple Counter in ASU VPL</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Exercise 2-4 Using While-Loop to implement Counter</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Exercise 2-5 Counter Activity in ASU-VPL</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Exercise 2-6 Vending Machine</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Exercise 2-7 Garage Door Opener</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Exercise 2-8 Party Generation</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>Section 3 Controlling EV3 Robots Using ASU-VPL</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Exercise 3-1 Build Your EV3 or Edison Robot</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Exercise 3-2 Drive-by-wire in ASU VPL</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Exercise 3-3 Connecting your robot to computer via Wi-Fi or Bluetooth</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Exercise 3-4 Improving the driving experience</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Exercise 3-5 Finding sensor reading from EV3 Brick (for EV3 only)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Exercise 3-6</td>
<td>Finding sensor reading by a program</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Exercise 3-7</td>
<td>Testing more sensors</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Exercise 3-8</td>
<td>Line-follower in ASU-VPL (if color/light sensor installed)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Exercise 3-9</td>
<td>Wall-following maze navigation (main diagram)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Exercise 3-10</td>
<td>Write the Init Activity</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Exercise 3-11</td>
<td>Write the Left1 Activity</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Exercise 3-12</td>
<td>Write the Right90 Activity</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Exercise 3-13</td>
<td>Write the Left90 Activity</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Exercise 3-14</td>
<td>Write the Right1 Activity</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Exercise 3-16</td>
<td>Write the Backward Activity</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Exercise 3-17</td>
<td>Write the ResetState Activity</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Exercise 3-18</td>
<td>Write the Forward Activity</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Exercise 3-19</td>
<td>Implementing the two distance algorithm in ASU-VPL</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Section 4 Edison Robot Programming**

<table>
<thead>
<tr>
<th>Exercise 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exercise 2</td>
</tr>
<tr>
<td>Exercise 3</td>
</tr>
<tr>
<td>Exercise 4</td>
</tr>
<tr>
<td>Exercise 5</td>
</tr>
<tr>
<td>Exercise 6</td>
</tr>
<tr>
<td>Exercise 7</td>
</tr>
<tr>
<td>Exercise 8</td>
</tr>
<tr>
<td>Exercise</td>
</tr>
<tr>
<td>----------</td>
</tr>
<tr>
<td>Exercise 9</td>
</tr>
<tr>
<td>Exercise 10</td>
</tr>
</tbody>
</table>