Abstract:
Designing a robot for the Trinity College Fire Fighting Home Robot Contest is a lengthy process that requires many hours and the control program cannot be debugged until late in the process after the hardware is built. The purpose of this simulator is to provide an easy quick way to design and test robot control programs allowing the team to efficiently create a design and build a winning robot.
ACKNOWLEDGEMENTS

Thank you to Dr. Donald Roberts for his help with both the math and programming side of this project.

I would also like to thank Mark Randall for all his help on the technical aspects of the robot.

LIST OF FIGURES

Figure 1: Object Relationships

Figure 2: The Three Major Parts

Figure 3: Excerpt from getA2D

Figure 4: Geometry of Movement

Figure 5: Arena Builder in Action
INTRODUCTION

The purpose of the Trinity College Fire Fighting Home Robot Competition (TCFFHRC) is to create autonomous robots that are socially relevant. The University of Evansville enters into this competition each year. The team that designs and builds the robot spends a lot of time and resources on this. Thus a simulator that allows a mockup of the robot to be tested increases the efficiency in the team’s use of time.

In the simulator, the user will be able to create an arena complete with walls, doorways, and candles. The user also will be able add a control program that will interact with the sensors and motors as if they were the real thing. This simulator will to test the core goal of the contest, putting out the candles.

BACKGROUND AND PROBLEM STATEMENT

The Trinity College Firefighting Home Robot Contest (TCFFHRC) “is an open, nonprofit event that requires invention of autonomous, socially relevant robots. The contests promote creativity, team work, the understanding and application of STEM subjects, and the sharing of ideas. [2]” The TCFFHRC has three levels in it. Level one takes place in an eight-foot by eight-foot arena that contains four rooms. The walls are between twenty-seven and thirty-four centimeters high. The arena is made of plywood and painted black with white stripes marking the doorways. The challenge for this level is the robot must locate a candle and put it out. To progress to level two, level one must be completed at least once in under three minutes. Level two takes place in one of four possible arenas of the same dimensions and room count of the first level. However, this time the arena is furnished with rugs and wall decorations. Again the goal is to put out a candle, but this time in under four minutes. Level three ramps up the difficulty.
The arena is composed this time by two level-two arenas connected by a one meter hallway. The robot starts in what is called the first arena. The primary goal also changes to retrieve a baby doll and then put out a candle, both located in the second arena, and bringing the baby doll back to the starting point. Additionally, two other candles will be lit after the start. The first one is lit ninety seconds after the start, and the second is lit 120 seconds after the start, or thirty seconds after the first. The time it takes to put out these candles is appended to the time of the baby rescue. To add more challenge, dog obstacles are placed in the arena. These obstacles are just stationary toy dogs. If the robot pushes the dog obstacle over or moves past it in the same hallway, then the run fails. If the robot moves the dog more than one centimeter, a fifty second penalty is added. The winner is whatever team completes the most levels with the lowest time score [2].

The competition also has very strict rules on the design of the robot. Firstly the robot must be autonomous, meaning that there is no human interaction needed for the robot to complete the challenge. The robot must fit into a box with a thirty-one by thirty-one centimeter square base with wall height of twenty-seven centimeters. There is no restriction on weight or material. The division of the competition that the University of Evansville team competes in is required to use a sound activation system. The microphone must be located at the top of the robot, it must have a blue background, and it must be labeled as “MIC” in contrasting colors to the background. The sound activation has a frequency of 3.8 kHz, and the emitter device will be located about twenty-five millimeters away from the microphone. Additionally there must be a sound detection LED. If the robot can be started by a hand-clap the robot will be disqualified. There must be a bright red LED on a white background that acts as a flame detection indicator. A Kill Power Plug which cuts all power to the sensors, control, and drive must be include and
mounted on a bright yellow background. As of 2016, the TCFHRC requires a carrying handle to be secured to the robot, the handle must be able to support the robot while being carried. They also recommend that the switches and microphone be located on the handle assembly [2].

Because of the strict restriction on both the design of the robot and the challenges that it must overcome the team designing the robot has a lot to take into account. This makes the design phase fairly complex and lengthy. The testing of the mechanical and sensor features of the robot must be done after a significant portion is built. Additionally, the control program cannot be tested until very late in the initial build. Before they can even test the robot in an environment similar to the contest, they will have to build a real-life one. Hence, a simulator of the different arenas would be very helpful. It would allow them to quickly create an arena for testing. This would allow the control program to be written and tested early in the design process. A few example results of this testing would be discovering that the sensor’s data is not being interpreted correctly, the control program is not correctly moving the robot, or the robot is unable to locate the flame. Normally, these problems could be caused by a fault in the control program, a fault in the wiring, or a number of other things, however the simulator will allow the team to determine if it is a fault in the control program because it will eliminate any mechanical faults. With the simulation the team will be able to rapidly create and test multiple iterations of the control program before committing to a winning design.

REQUIREMENTS AND SPECIFICATIONS

The largest part of this project is getting the robot to act like a real one. Thus initially the simulated robot has fixed wheel placement, a standard motor for movement, and base design.
The robot initially is fixed at eight inches wide, and eight inches long. All the sensors are based on their real life counter parts. This means that they take realistic readings and communicate with the control program just as the real ones do. There are four types of sensor that included in this project. The first is a touch sensor. The touch sensor is used for course correction and determining if they are bumping into wall. The next sensor is the Infrared Sharp Sensor. This is a range-finding sensor. The UV Tron is a light detecting sensor that detect levels of UV lights. The values that are returned are in the form of pulses of the same magnitude, the intensity of light is given by the frequency of pulses. The last sensor is the line board. It detects a painted line on the ground.

The arena creation tool is the next big part of this project. This tool allows the team to create different layouts to model the actual competition. The walls themselves do not allow the robot to pass through them, and they cause collisions with the range finding sensors and the touch sensors. The light sensor is not be able to detect light levels from the other side of a wall. Thus it acts as a real wall. There is a tool to designate doorways allowing for the full creation of rooms. There is a tool to add one or more candles to the arena. There also is a tool to arbitrarily place the start position for the robot. This allows the team to estimate to how their robot is performing. The non-required goals are tools to add the static dog obstacles and rugs.

DESIGN APPROACH

The objects in this project all inherit from the Basic Figure class provided by SmallDraw, and either implement the IArenaObj or the IRobotObj interfaces. All objects in the arena use interfaces that make them detectable by certain sensors. These interfaces are: IDistance, ISolid, IFire, and IFlat. This relationship between sensors, objects and interfaces are fully fleshed out in Figure 1. Each sensor detects a different interface allowing for other IArenaObj to be added to
the simulation simply by implementing the appropriate interface and without the need to edit the sensors. The interfaces reflect different properties of the object implementing them.

Figure 1: Object Relationships

The simulation can be functionally decomposed into three parts as shown in Figure 2. The first is the user interfaces for the Arena Builder. The second part is the Simulation. And the third part of this is the Parser which functions as the Application Program Interface (API). The API is the part of the program that will transfer data between the robot and the control program.
SmallDraw in C# [1] is a third party piece of software that is a geometric shape drawing application. It is used as the basis the user interface because it is a simple program that can be easily modified to meet the graphical requirements for the simulation. The user interface for the arena builder is graphical and allows the user to place various objects inside of it. The objects are walls, doorways, candles, and the robot start space. The Arena Builder has two important parts. The _canvas is where all the drawing occurs, and it stores all the objects drawn on it in _figures. The attribute _figures is an IList of IFigures that is passed from the Arena Builder’s _canvas to the Simulator’s _canvas to allow the Simulator to duplicate Arena Builder’s _canvas. Arena Builder’s _groupbox contains the buttons that create the Arena Objects.

Graphically the wall are represented by a thick black line. The walls also cause a collision with the robot and sensors. This is done by implementing the ISolid interface. Walls also implement the IDistance interface which allows the range finding sensors to detect them and return a value. The wall’s identifying attributes will be two ordered pairs that are start and end. This will allow for easier collision detection. Doorways are graphically represented as a blue line. The doorways implement only the IFlat interface. This makes the Line Sensor detect it.
Like the wall doorways will have a start and end. The candle objects implements the ISolid, IDistance, and IFire interfaces. It is graphically represented by a red dot. The robot start position implements only the IFlat interface. It is graphically represented by a green circle.

```csharp
foreach(IDistance f in _canvas FiguresOfType<IDistance>())
{
    temp = Util.Geometry.LineIntersection(this.lazer.getStart(), this.lazer.getEnd(),
                                             f.GetStart(), f.GetEnd());
    if (!temp.IsEmpty)
    {
        _points.Add(new PointF(temp.X, temp.Y));
    }
}
```

Figure 3: Excerpt from getA2D

The robot object contains all the sensors in an IList of IRobotObj called _sensors. All of the sensor calls from Parser are passed to the robot, and from there they are passed to their respective sensor. The Range Finder is modeled with the function getA2D(char select). When this method is called the robot uses the parameter select to determine which Range Finder is being queried and returns the value from the proper instance. It does this by first finding all the objects that implement the IDistance interface shown in Figure 3. This gathers all the valid objects that intersect with the Range Finder’s lazer. After this code executes getA2D sends _points to lazer.firstWall which returns a double representative of the distance in pixels. The method firstWall first checks to see if _points is empty, returning the farthest measurement. Otherwise it calculates the distance from the start of the range finder with each point of intersection and compares them. The UV Tron is an orange square centered in the middle of the robot figure. It has two method: checkForFire and scanForFire. The method checkForFire takes
no parameters and returns a Boolean based on whether or not there are any objects in the way. ScanForFire takes a floating point “facing” as a parameter. Facing is the angle at which the robot is facing. ScanForFire returns the angle of an object implementing the IFire interface. The returned angle is between 0 degrees and 180 degrees and -1 if the fire is not detectable or if it is beyond 180 degrees. The Line Board has a detectLine method that will return a value of 1 if a line is under it

The Simulator handles all the animation, and movement of the robot. Its animation runs on a timer that fires every 5 milliseconds. The algorithm for movement is as follows:

1) The angle at which the robot is facing in radians is calculated
2) The left and right motor speeds are checked to see if they are both in the stopped status, if they are then the movement vector is set to zero and the returned angle is set to zero.
3) The left and right motor speeds are calculated in pixel measurements
4) Then the faster of the two motors is determined and a flag is set to true if the right side is faster, and false if it is not
5) The angle theta is calculated using the arctangent of the fast motor minus the slow motor divided by the axle length.
6) The distance moved is calculated by taking the tangent of theta and then multiplied by half the axle length.
7) After the robot is moved, it rotates by theta degrees.
The Parser is the tool that contains all the calls that the control program makes to the robot, as well as the control program itself. The control program was originally supposed to be written in C however this was changed to C# because while C# is object-oriented, C is not, making the two languages incompatible. Every method in the Parser passes the call along to the robot or in the case of movement to the Simulation. The Parser is formatted to include as little access as possible because it is the file that the control program is copied into and thus comes into contact with the user the most.
The other design that was considered was the idea of having a “Super Arena Object”. Instead of using interface to represent properties, the super object would have boolean attributes of similar names, solid instead of ISolid, distance instead of IDistance, etc. The best way of implementing this would be to add these attributes to the BasicFigure provided by SmallDraw. The decision to use interfaces was based on ease to create more arena objects by just adding the proper interfaces and the searching mechanic that is used.

RESULTS

The simulator works in a very simplistic fashion. The Arena Builder allows the user to create different layouts, and walls and doorways display their length in real time as they are moved about. The sensors return an accurate representation of real world data. The Simulator animates correctly. The Parser works as intended.

The Arena Builder shown in Figure 5, allows for the user to create the room layouts in whatever configuration is needed. The canvas itself is a representation of an eight foot by eight foot square set out by the rules of the competition.
CONCLUSION

This project will assist the team in creating a control program that is free of logic errors, limiting the number of possible issues that will come up in testing the real robot. Future work for this project could include adding more sensors or obstacles. Additionally a more realistic representation of physics could be added to the motion.
REFERENCES:


BIOGRAPHY

Evan DeCraene grew up in Chicago playing Dungeons and Dragons with his father and brother. He has been a Boy Scout all his life and even attained the rank of Eagle Scout. After graduation he hopes to move back to Chicago and get a job in the field of cybersecurity.