Admin Manual

I. Introduction to the IRET

Our team, Team H, is very pleased to hand this product off to you. We're proud of what we've managed to build on top of the already existing application, built by the preexisting Team L. The previous team set up a React application and made the application responsive using Bootstrap and CSS Media Queries, and fully implemented the differential drive, bicycle, and tricycle motion models in JavaScript and visually layed out the paths of the motion models and the motion models themselves using the Canvas API. The motion models have start, stop, and reset functionality implemented, as well as functionality to allow the motion models to change on-demand given user input. They also fully implemented the pathfinding algorithm RRT, or Rapidly Exploring Random Trees, complete with arbitrary obstacles that could be drawn by the user and start, stop, reset, and 1-step buttons that implement said functionality. We managed to add on the Bug0 pathfinding algorithm, which uses a lot of the obstacle drawing from the previous team. Unlike RRT, it follows a wall of an obstacle in order to get to a goal, which is functionality we implemented ourselves.

In addition to those things, we also implemented ranges for values given by the user, implemented unit tests for individual React components, and ensured that drawings using the Canvas API could not go off-screen. We ensured to make the application as user-friendly as possible while also being informative as possible, sticking to the plan we initially set at the beginning of the semester.

To access all these amazing things, here is the link to our GitHub repo. To set up and run the application on your local machine, a client video was made so you guys can run and modify the project on your own.

II. Project Dependencies and Documentation

There are several libraries and dependencies that we installed and used for this application and we want to inform you of. One of the dependencies that we installed for the application is something called Snowpack. Snowpack is a modern, lightweight build tool for faster web development. Since most build tools for React bundle the application even during the development phase, thus greatly slowing down the process of coding, we decided to use snowpack to bundle and run our application. In addition to Snowpack, we also used Bootstrap. Bootstrap is an open source CSS toolkit that makes applications responsive and faster. Finally, in addition to Bootstrap, we installed Jest and Enzyme. These are testing tools that allow you to

run unit tests on React Components and even run tests on component props and component state, something very important with React.

In addition to installable dependencies, we used the Canvas API, which is automatically built into HTML and JavaScript. We also used JQuery, a JavaScript library to make HTML documentation traversal and manipulation easier and faster. The entire application is written using HTML, CSS, JavaScript, and Python. Here is a formal list of the dependencies and their documentation if you would like to look into them.

Snowpack: https://www.snowpack.dev/tutorials/react

Bootstrap: https://getbootstrap.com/

Jest: https://jestjs.io/docs/tutorial-react

Enzyme: https://enzymejs.github.io/enzyme/

Canvas API: https://developer.mozilla.org/en-US/docs/Web/API/Canvas_API

JQuery: https://jquery.com/

III. Project Setup

Local Setup

Below are the steps to run the project locally to ensure the proper implementation of the Interactive Robotics Education Tool. This will also be useful if our client wants to continue building on top of the code.

1. Prepare the necessary tools such as logging onto your GitHub account and updating your Visual Studio Code (or any other IDE of your choosing) to the newest version. To create a GitHub account, follow the steps listed at https://github.com/signup. To download the newest version of Visual Studio Code, please visit https://code.visualstudio.com/download and follow the instructions for your chosen operating system.
2. Navigate to our project repository at https://github.com/zuntue/robotedu on the "main" branch.
3. Clone the repository into Visual Studio Code. To do so, first navigate to Visual Studio Code and open a terminal by selecting "Terminal" in the upper left hand of the screen and then selecting "New Terminal" in the dropdown menu. Then within the new terminal, run the command `**git clone https://github.com/zuntue/robotedu.git**` in the terminal and hit enter. Follow the steps in order to place the code within a local folder on your device. You can also find more directions to clone the repository by selecting the green "Code" dropdown button on our repository within GitHub.
4. After cloning the repo, Visual Studio Code will automatically move to the directory and feature the files in the left hand section under the "Explore" tab. If this step does not

automatically happen, run the command **`cd robotedu`** in the terminal to move to the directory.

5. Within the same terminal as before, run the command **`npm install`** to install the dependencies we used in order to create the project.
6. To locally display the web app, run the command **`npm start`** within the terminal. This will locally host the app for you to interact with and test.

Deployment

Our client wants to be able to incorporate our web app into their existing robotics website. The instructions on how to do so are described below. Make sure to have first run the project locally, following the steps outlined above, to ensure all dependencies are installed.

1. Navigate to the `add-static-file-build-command` branch by running **`git checkout add-static-file-build-command`** in the Visual Studio Code terminal. This branch is different from the `main` branch in that it is used to build our project for deployment.
2. Add a `homepage` field to the `package.json` file. Using our coach Louie Lu's server, we have attached a screenshot with the line in question highlighted of what this step should look like below. Simply replace Louie's domain page to yours.

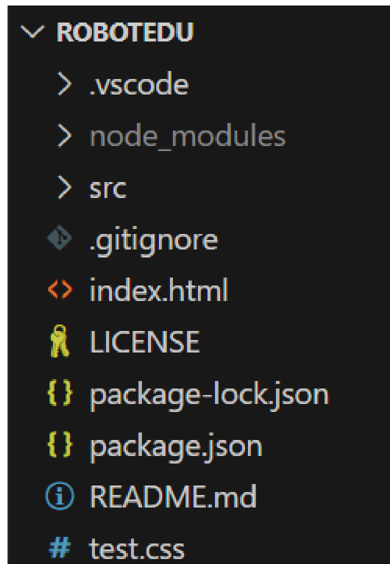
```
{} package.json > ...
{
  "name": "robotedu",
  "version": "0.1.0",
  "main": "index.js",
  "scripts": {
    "start": "snowpack dev",
    "test": "react-scripts test"
  },
  "homepage": "https://static.louie.lu/build",
  "author": "",
  "license": "MIT",
  "dependencies": {
    "@testing-library/jest-dom": "^5.16.3",
    "@testing-library/react": "^12.1.4",
    "@testing-library/user-event": "^13.5.0",
    "enzyme": "^3.11.0",
    "jest": "^27.5.1",
    "jquery": "^3.6.4",
    "react": "^17.0.2",
    "react-dom": "^17.0.2",
    "react-router-dom": "^6.2.2",
    "react-scripts": "5.0.0",
    "snowpack": "^3.8.8",
    "web-vitals": "^2.1.4"
  },
  "jest": {
    "transform": {
      "^.+\\.(js|jsx)$": "babel-jest"
    }
  },
  "devDependencies": {
    "@wojtekmaj/enzyme-adapter-react-17": "^0.6.7"
  }
}
```

3. Run **`npx react-scripts build`** in the terminal to build a static file.
4. Upload the static file to your server.

5. Use the <iframe> element in your HTML code to embed the static file into the webpage of your choosing.

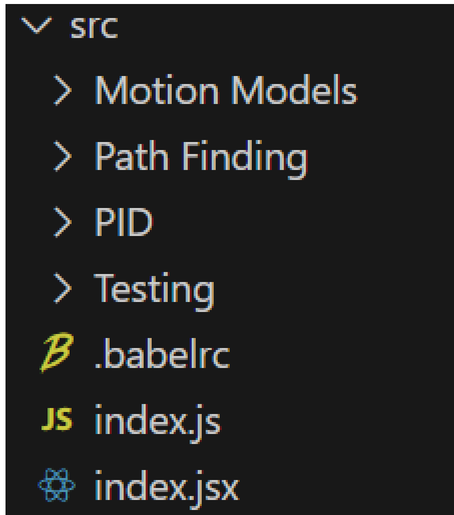
IV. Project Architecture

This is the structure of our entire application.

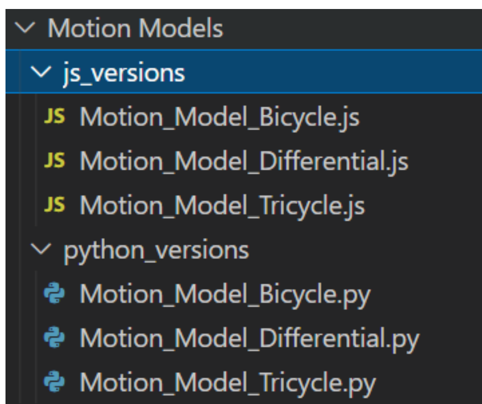


As you can see, we have several folders, the most important being src. Outside any folder are the index.html and test.css files. The test.css file governs the CSS of the application apart from Bootstrap CSS which is already pre-installed in node_modules and the index.html file is the "root node" of the application and is where the document itself begins. However, it is unlikely you will change the index.html file because the app is really located inside the src folder. You can essentially ignore package.json, package-lock.json, and the .gitignore file. The README.md file is just a text file that shows up on our GitHub repo's page.

Now, this is where the magic happens. Inside the src folder, there is a .babelrc file (you can ignore this, this is only for testing purposes), a folder called Testing, and an index.jsx file. The index.jsx file contains ALL OF THE RUNNING CODE FOR THE APPLICATION. Every time you run npm start in a terminal, any change you make in index.jsx will be immediately rendered on your screen.



Inside the Motion Models folder, we have separated files into python_versions and js_versions which are the names of the two folders. Initially, when we receive a pseudocode, we translate it into Python. Then, after testing to make sure the code works in Python, we immediately translate the same code into JavaScript and this JavaScript code is the thing we import into the app.



Inside the Path Finding folder, there is a folder called Tree_Struct, which contains a file converted to JavaScript that constructs a tree, and the pathfinding algorithm files, which are directly translated from pseudocode to JavaScript. Currently, the previous team set up RRT, and we managed to build a fully functioning Bug0. There are also files for Bug1 and Bug2 already set up, and already in the project folder.

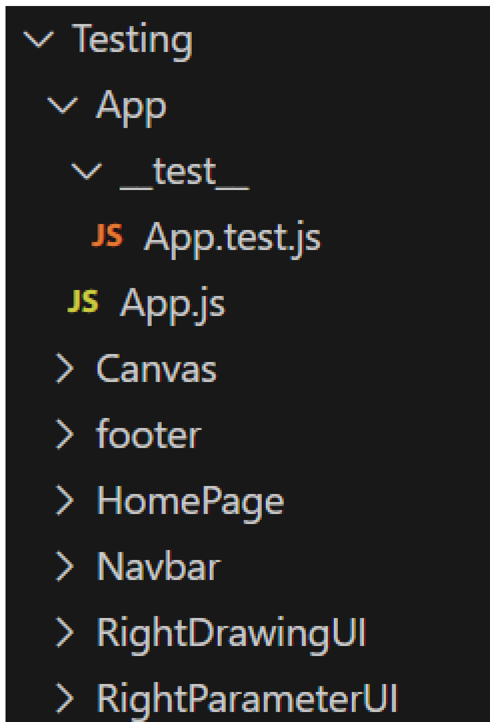
Path Finding
Tree_Struct
treeAll.js
bug0class.js
bug1class.js
bug2class.js
RRT.js

Inside the PID folder, you will find three essential files that comprise the PID controller visualization tool: PID_testing.html, PID_testing.js, and PID_testing_docs.md. The PID_testing.html file serves as the user interface, presenting an interactive layout for users to input parameters and view the controller's output. PID_testing.js contains the JavaScript implementation of the PID controller algorithm, directly translating the underlying principles into a functional and interactive experience for users. Finally, the PID_testing_docs.md file contains the PID Controller Testing Guide which provides comprehensive documentation, including a detailed explanation of the PID controller's input and output fields, as well as step-by-step instructions and tips for using the visualization tool effectively. Together, these files form the foundation of the PID controller module, enabling users to explore and understand this vital control mechanism in a hands-on manner.

Currently, this aspect is not fully implemented into the IRET. The navbar will take you to a different page where you will be able to access the PID controller. In the future, this should be completely implemented into the official tool.

PID
PID_testing_docs.md
PID_testing.html
PID_testing.js

Inside the Testing folder, we have each React Component separated into its own folder. Within each component folder, there is another folder called __test__ which contains the test.js file that tests each component and the component itself in the component folder. This is the entire structure of our application.



V. Adding to the Project

Here are instructions on how to add to the project in the future in order to create new visualizations. First of all, we don't really recommend changing anything in the index.html file as that could break the application because there are complex dependencies and libraries (previously mentioned) that are imported into these files. We also don't really recommend changing anything in the Testing folder unless you would like to add more unit tests on your own. To start, if you have the pseudocode for a motion model or a pathfinding algorithm, this must first be translated into a programming language. The first step is translating pseudocode into a programming language like Python since Python is very good with mathematics.

The next step is to translate the Python program into a JavaScript program. You can achieve this with the use of the *class* keyword and by making a class you're going to use in the application.

```
const differential = new diff(leftWheelRadius, rightWheelRadius, distBetweenWheels, leftAngularVelocity, rightAngularVelocity,
```

Finally, it's time to use the JavaScript program inside the index.jsx file. First, import the JavaScript program from either the Motion Model or the Path Finding folder.

```
import cycles from './Motion Models/js_versions/Motion_Model_Bicycle';
import tricycle from './Motion Models/js_versions/Motion_Model_Tricycle';
import diff from './Motion Models/js_versions/Motion_Model_Differential';
import RRT from './Path Finding/RRT';
import Bug0 from './Path Finding/bug0class'
import Bug1 from './Path Finding/bug1class'
import Bug2 from './Path Finding/bug2class'
```

Then, for motion model additions, you need to edit the following areas in the index.jsx file.

- In the App component, you need to add each parameter for the motion model inside this.state.

```
constructor(props) {
  super(props);
  this.state = {
    page: '',
    steeringAngle: 0,
    angularVelocity: 0,
    distBetweenWheels: 0,
    leftAngularVelocity: 0,
    rightAngularVelocity: 0,
```

- Then, after the App component's constructor, you need to add data binders for each new parameter since React does not accomplish two-way data binding on its own. Just follow the pattern in the code.

```
this.handleSteeringAngleChange = this.handleSteeringAngleChange.bind(this); //steer
this.handleDistFrontToBackChange = this.handleDistFrontToBackChange.bind(this); //d
this.handleAngularVelocityChange = this.handleAngularVelocityChange.bind(this); //a
this.handleFrontWheelRadiusChange = this.handleFrontWheelRadiusChange.bind(this); /
this.handleDistBackTwoWheelsChange = this.handleDistBackTwoWheelsChange.bind(this);
```

- In the toggleResetParameters() and togglePage() functions in the app component, you will have to do the same thing in the first bullet point where you set the state of each new parameter to 0.
- Then, for each data binder for each parameter, you need to define the data binder function, which also goes in the App component.

```
handleDistBetweenWheelsChange = (num) => {
  this.setState({ distBetweenWheels: num }, () => {
    console.log('');
  });
}

handleLeftAngularVelocityChange = (num) => {
  this.setState({ leftAngularVelocity: num }, () => {
    console.log('');
  });
}
```

- Scroll down to the switch statement in the render() method at the end of the App component. Copy a case like ‘Diff. Drive’ all the way including its break statement and add a new case to the switch statement like ‘MOTION-MODEL-NAME’. Then, follow the patterns in the code to add new attributes for each parameter in the motion model. ONLY EDIT THE PARAMETER NAMES, DON’T EDIT THE JQUERY ATTRIBUTES OR THE TOGGLERESETPARAMETERS ATTRIBUTE FOR THE LOWERCONTROLUI COMPONENT. For instance, if I had a new parameter like leftDoubleVelocity, I would delete the steeringAngle, angularVelocity, distFrontToBack, and frontWheelRadius attributes for the Canvas component and add leftDoubleVelocity = {this.state.leftDoubleVelocity}. I would do the same thing and add onLeftDoubleVelocityChange = {this.handleLeftDoubleVelocityChange} for the RightParameterUI component.

```
case 'Bicycle':
  return (<><Navbar togglePage={this.togglePage} /><Canvas jQuery={this.state.page}
    steeringAngle={this.state.steeringAngle}
    angularVelocity={this.state.angularVelocity}
    distFrontToBack={this.state.distFrontToBack}
    frontWheelRadius={this.state.frontWheelRadius}
  /><RightParameterUI
      onAngularVelocityChange={this.handleAngularVelocityChange}
      onSteeringAngleChange={this.handleSteeringAngleChange}
      onFrontWheelRadiusChange={this.handleFrontWheelRadiusChange}
      onDistFrontToBackChange={this.handleDistFrontToBackChange}
      jQuery={this.state.page} /><LowerControlUI jQuery={this.state.page} toggleResetParameters={this.toggleResetParameters} /></>)
  break;
```

- In the Navbar Component, add the new component. The name attribute in the HTML element is very important so make sure it matches the text.

```
<ul class="dropdown-menu">
  <li><a href="#Model_1" onClick={this.togglePage} name="Diff. Drive">Differential Drive</a></li>
  <li><a href="#Model_2" onClick={this.togglePage} name="Bicycle">Bicycle</a></li>
  <li><a href="#Model_3" onClick={this.togglePage} name="Tricycle">Tricycle</a></li>
</ul>
```

- Then in the Canvas Component, add another JQueryCode method like JQueryCodeMOTION-MODEL-NAME and copy the code within either JQueryCodeBicycle or JQueryCodeTricycle and paste in the new JQueryCode method. Then, you can edit the variables to match those in the JQuery code. In the JQueryCode method, this is where you are actually drawing on the canvas.

```
var leftWheelRadius = this.props.leftWheelRadius
var rightWheelRadius = this.props.rightWheelRadius
var distBetweenWheels = this.props.distBetweenWheels
var leftAngularVelocity = this.props.leftAngularVelocity
var rightAngularVelocity = this.props.rightAngularVelocity
```

- Then in the ComponentDidMount() and componentDidUpdate() lifecycle methods, follow the pattern like so.

```
componentDidMount() {
  switch (this.props.jQuery) {
    case "RET":
      this.jQueryCodeRET();
      break;
    case "Bug0":
      this.jQueryCodeBug0();
      break;
    case "Bug1":
      this.jQueryCodeBug1();
      break;
    case "Bug2":
      this.jQueryCodeBug2();
      break;
    case "Diff. Drive":
      this.jQueryCodeDiffDrive();
      break;
    case "Bicycle":
      this.jQueryCodeBicycle();
      break;
    case "Tricycle":
      this.jQueryCodeTricycle();
      break;
  }
}
```

```
componentDidUpdate() {
  switch (this.props.jQuery) {
    case "RET":
      this.jQueryCodeRET();
      break;
    case "Diff. Drive":
      this.jQueryCodeDiffDrive();
      break;
    case "Bug0":
      this.jQueryCodeBug0();
      break;
    case "Bug1":
      this.jQueryCodeBug1();
      break;
    case "Bug2":
      this.jQueryCodeBug2();
      break;
    case "Bicycle":
      this.jQueryCodeBicycle();
      break;
    case "Tricycle":
      this.jQueryCodeTricycle();
      break;
  }
}
```

- Scroll down to the RightParameterUI component and follow the pattern to define the data binders in the child component. THIS IS WHERE YOU SET THE BOUNDS FOR EACH PARAMETER.

```
class RightParameterUI extends React.Component {
  constructor(props) {
    super(props);

    this.handleSteeringAngleChange = this.handleSteeringAngleChange.bind(this);
    this.handleAngularVelocityChange = this.handleAngularVelocityChange.bind(this);
    this.handleFrontWheelRadiusChange = this.handleFrontWheelRadiusChange.bind(this);
    this.handleDistFrontToBackChange = this.handleDistFrontToBackChange.bind(this);
    this.handleDistBackTwoWheelsChange = this.handleDistBackTwoWheelsChange.bind(this);
    this.handleDistBetweenWheelsChange = this.handleDistBetweenWheelsChange.bind(this);
    this.handleLeftWheelRadiusChange = this.handleLeftWheelRadiusChange.bind(this);
    this.handleRightWheelRadiusChange = this.handleRightWheelRadiusChange.bind(this);
    this.handleLeftAngularVelocityChange = this.handleLeftAngularVelocityChange.bind(this);
    this.handleRightAngularVelocityChange = this.handleRightAngularVelocityChange.bind(this);
  }
  //handling limits on parameters inputted by the user
  handleDistBackTwoWheelsChange(e) {
    if (e.target.value <= 30 || e.target.value > 200) {
    } else {
      this.props.onDistBackTwoWheelsChange(e.target.value);
    }
  };

  handleDistFrontToBackChange(e) {
    if (e.target.value <= 10 || e.target.value > 100) {
    } else {
      this.props.onDistFrontToBackChange(e.target.value);
```

- In the RightParameterUI's render method, add another case in the break statement where you add the new parameters for the motion model.

```
case 'Diff. Drive':
  return (<div id="rightParameterUI">
    <h4>Parameters</h4>
    <h5>Robot Properties</h5>
    <label for="leftWheelRadius">Left Wheel Radius (0 &#60; x &#8804; 10)</label>
    <br></br>
    <input type="number" id="leftWheelRadius" placeholder='0' onChange={this.handleLeftWheelRadiusChange}></input>
    <br></br>

    <label for="rightWheelRadius">Right Wheel Radius (0 &#60; x &#8804; 10)</label>
    <br></br>
    <input type="number" id="rightWheelRadius" placeholder='0' onChange={this.handleRightWheelRadiusChange}></input>
    <br></br>
```

- Finally, scroll up to toggleResetParameters() in the LowerControlUI component. For each input id you made in RightParameterUI, clear the input.

```
toggleResetParameters() {
  this.props.toggleResetParameters()
  switch (this.props.jQuery) {
    case 'Diff. Drive':
      document.getElementById('leftWheelRadius').value = '';
      document.getElementById('rightWheelRadius').value = '';
      document.getElementById('distBetweenWheels').value = '';
      document.getElementById('leftAngularVelocity').value = '';
      document.getElementById('rightAngularVelocity').value = '';
      break;
```

If a future team is looking to fully finish Bug1 or Bug2, or another Pathfinding Algorithm, I would recommend looking through the implementation of Bug0. Both of them will follow Bug0's implementation as they are, and seeing how we accomplished Bug0 will assist in figuring out

how to implement wall-following in Bug1 and Bug2. Looking through the RRT file and its jQuery code in index.jsx and understanding how they interact would help as well.

VI. Thank you.

We as a group give you our biggest thanks for remaining open and communicative with us throughout the semester and giving us your actual feedback through our meetings with you on how our project looked and felt. We genuinely appreciate being open with us about not only the expectations about the requirements but the implementation of those requirements and how we could do the project whichever way we felt. As a group, it felt amazing to not have many restrictions in implementing the requirements. We really hope that you can add to this project and host this project online in the future and we hope you as our clients are satisfied with our work.

Thank you, Janine Hoelscher and Ron Alterovitz.

***Disclaimer: Because we worked on legacy code, we thought it best to add information regarding our updates in the already existing admin manual.