Project Report: PS-1 VECC

Souradeep Chakraborty
Oytrik Das
Rijul Ganguly
Ashutosh Mishra

Swarm Robotics Team

Souradeep Chakraborty
Oytrik Das
Rijul Ganguly

Overview

> Orientation

> Phase 1-4

- 1: Setup/Hands-on
- 2: WeBots familiarization
- 3: Single Robot Coverage
- 4: Establishing Communication

> Final design

> Upcoming Milestones

Orientation

Familiarization with problem statement

- The problem statement assigned was “Swarm Robot Area Coverage algorithm modeling and simulation”.
- In order to familiarize ourselves with the field of study systematic literature review was conducted.

Systematic Literature Review

- Various groundbreaking research papers in this field were studied ranging from single robot coverage systems to “Voronoi partition coverage using Lloyd’s algorithm in non-convex environments”.

Phase 1 - Setup

A virtual working environment

- Ubuntu 16.04 Xenial was installed as a virtual machine on a Windows 10 host
- Anaconda was used to set up a Jupyter Notebook framework to work with Python.

WeBots

- Initial Setup of WeBots was done for future simulation.

MATLAB

- Followed through a series of MATLAB codes by PhD student - Mr. Subhojit Indra, which modeled single robot coverage algorithms.

Phase 2 - WeBots Tutorials & Familiarization

Followed through all the official Tutorials in the official WeBots page

- Each tutorial helped familiarize ourselves with WeBots
- At the end of the series we modeled dynamic objects in WeBots with an e-Puck robot designed to avoid collisions with anything in the environment (which comprised of 4 walls and the dynamic objects like balls, dumbbells etc)

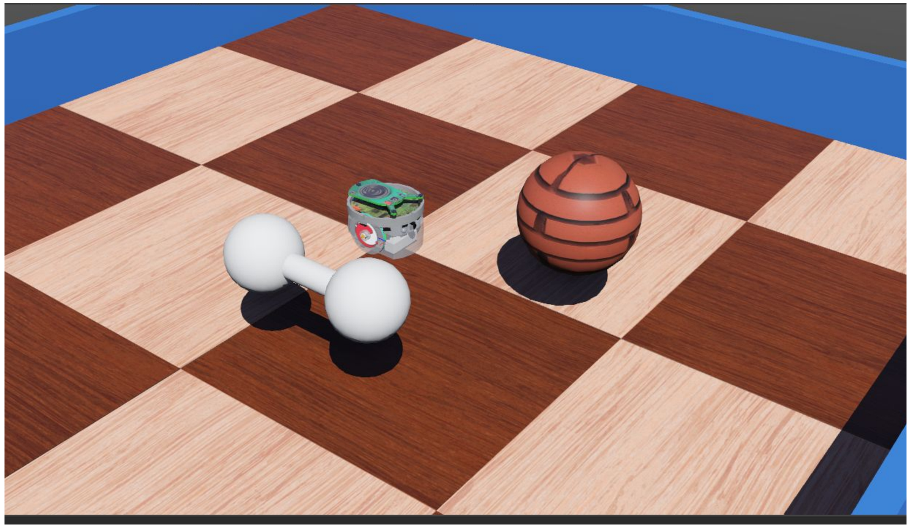


Figure: End result of Tutorials - this kind of a world was created, the small circular robot in the centre is designed to move randomly throughout the confines of the 4 blue walls while avoiding collisions with the ball and the dumbbell

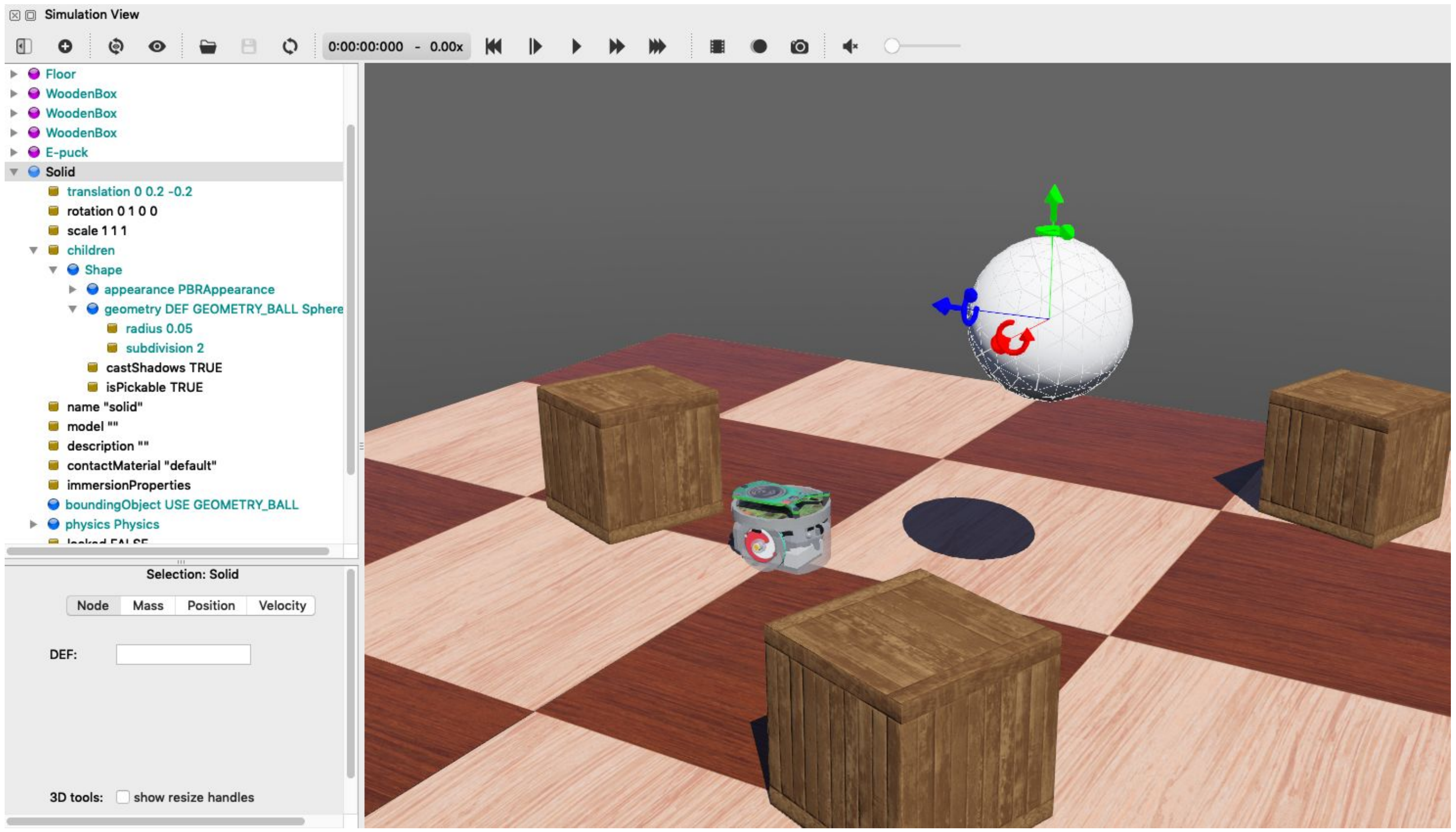
Phase 2 - WeBots Tutorials & Familiarization

Read through the other Documentations and Forum posts for further experience

- The tutorial series of WeBots is in no way complete, and as such, reading the Documentation was necessary to understand intricate details like Proto nodes, Controller logic, Communication and Supervisors.
- WeBots has had approximately 1k users in the last year which makes it very hard to find common issues online. Even the Documentation of WeBots is quite incomplete and as such, we resorted to Forums and Discord communities for searching issues we were facing.

Familiarization with Controller logic and PROTO coding

- Each e-Puck robot in WeBots is controlled using a controller logic that is written in C,C++,Java or Python (we are currently using C).
- Each element in WeBots (ranging from the floor in the environment to the e-Puck robot) has certain properties and sub-elements having their own properties and so on. This logic is maintained using PROTO codes that describe the hierarchical structure of each element in the environment. For example, we found out that the e-Puck can support two extension slots that need to be manually added in the PROTO code of the e-Puck.



```
▶ Floor
▶ WoodenBox
▶ WoodenBox
▶ WoodenBox
▶ E-puck
▼ Solid
    translation 0 0.2 -0.2
    rotation 0 1 0 0
    scale 1 1 1
  ▼ children
    ▼ Shape
      ▶ appearance PBRAppearance
      ▼ geometry DEF GEOMETRY_BALL Sphere
          radius 0.05
          subdivision 2
        castShadows TRUE
        isPickable TRUE
    name "solid"
    model ""
    description ""
    contactMaterial "default"
    immersionProperties
    boundingObject USE GEOMETRY_BALL
  ▶ physics Physics
    locked FALSE
```

Selection: Solid

Node	Mass	Position	Velocity

DEF:

3D tools: ☐ show resize handles

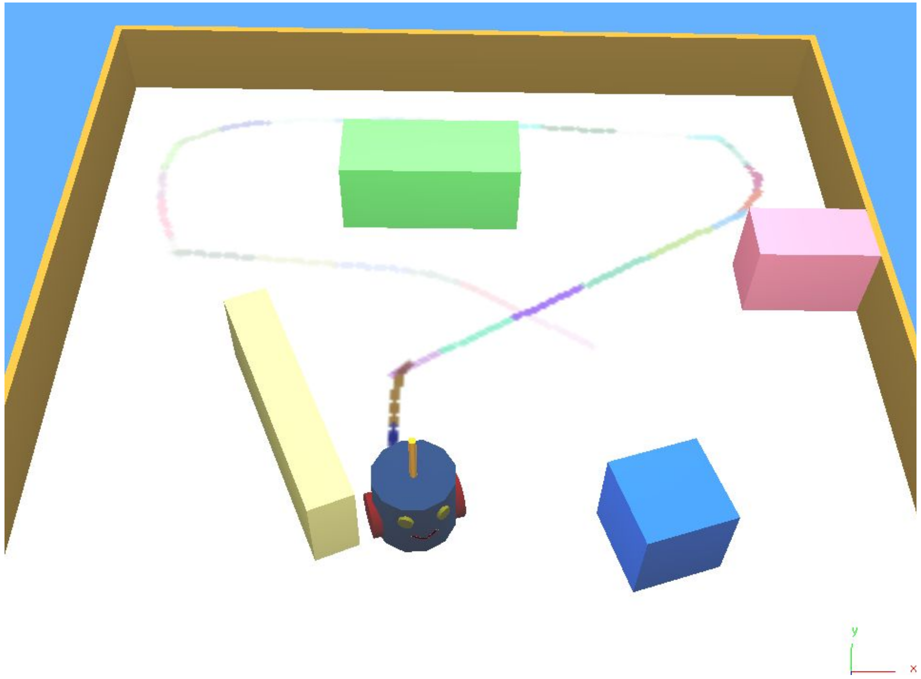
Figure: A typical environment in WeBots showing the WorldView pane in the left with details of PROTO Nodes and elements in the environment.

Phase 3 - Single Robot Coverage

Setting up Pen Tool

- In order to have a proof of concept of our coverage algorithm, we must first add a mechanism to find out which part of the map has been/is being covered by the e-Puck.

- This requires adding an extension slot to the e-Puck by modifying the PROTO Code to add a Pen Tool. The Pen tool essentially permanently colors the region of the floor visited by the e-Puck with a predefined customizable color.



Phase 3 - Single Robot Coverage

Zig-Zag coverage algorithm

- One of the fundamental single-robot coverage algorithm is the zigzag algorithm wherein the robot follows a straight line path in a zigzag manner.
- We implemented this algorithm on WeBots.
- To map the path followed by the robot we used the Pen tool.
- We assumed the start position to be a fixed corner, as shown in the adjacent figure.

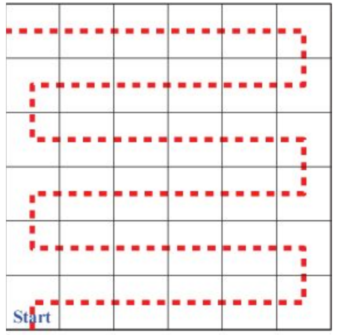


Figure: the Zigzag algorithm

Phase 4- Mapping Path & Communicating

GPS to continually retrieve positions

- Each e-Puck in WeBots comes with its own GPS, which is added to its TurretSlot. We will need this data later on to plan paths for robots in order to avoid collisions.
- In order to design a multi-robot coverage system, to achieve optimal coverage times, we must also ensure that the regions already mapped by robots must be saved somewhere

Alternate ways to get position

- A Supervisor Node can be added to find the positional coordinates. This will be helpful when we want a central logic for all the robots in the system. (Not really swarm robots oriented, which prefers a decentralized logic).

Phase 4- Mapping Path & Communicating

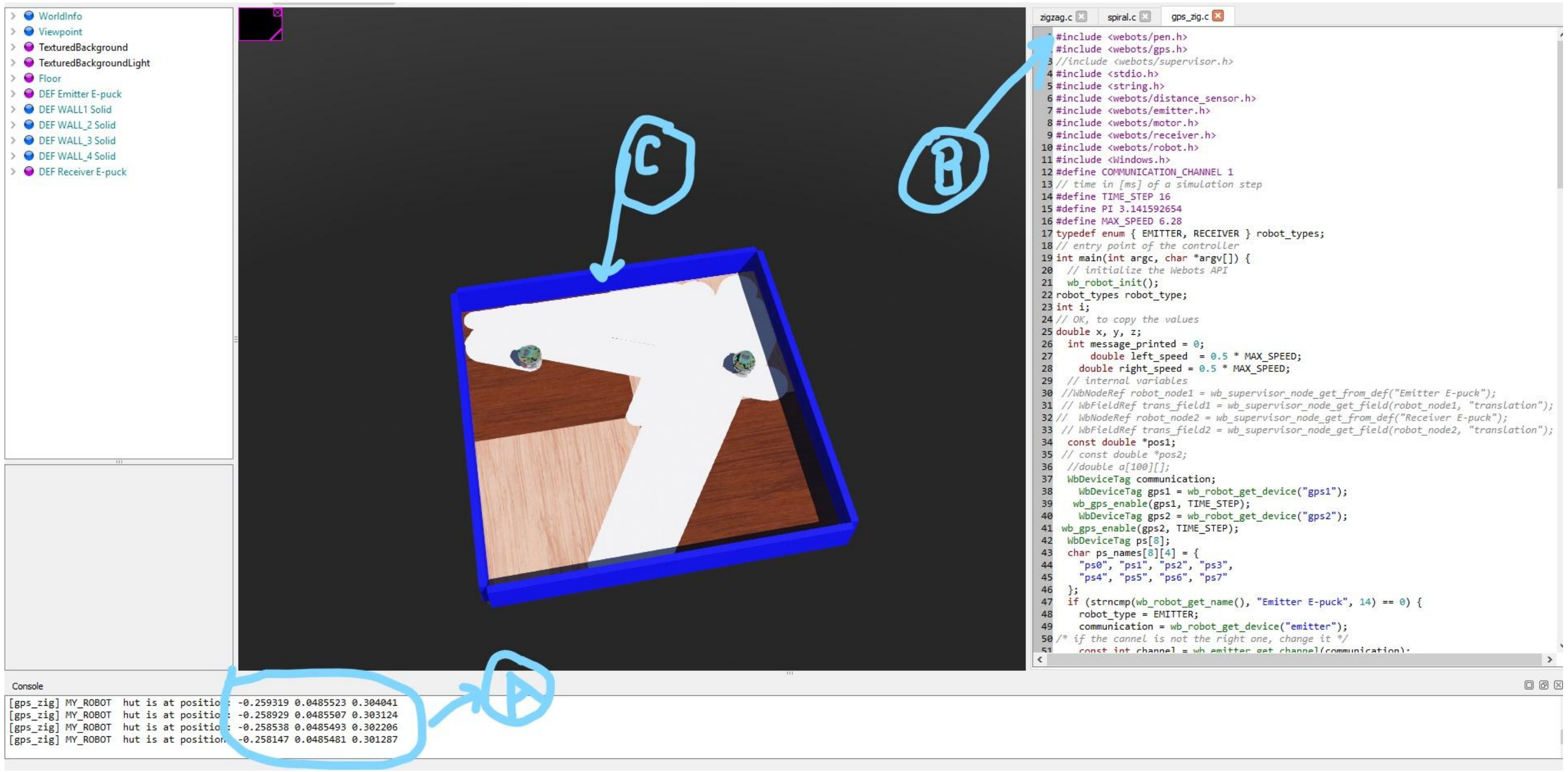
Communication

- The next part in achieving our goal is to establish a communication system between various e-Pucks. This can be done using Emitter and Receiver channels in the e-Puck.
- We modeled and simulated an environment with two e-Puck, wherein one e-Puck continually emits its position and the other e-Puck receives its coordinates, provided its in a communication range, and prints these coordinates on the console.

Integrating everything together

- Once we could establish communication between two e-Pucks, we integrated all previous modules together to finish the final model - a simulation environment with two e-Pucks following the zigzag algorithm continually communicating their coordinates with each other and mapping their path using the Pen tool.

Final Design



```c
#include <webots/pen.h>
#include <webots/gps.h>
//include <webots/supervisor.h>
#include <stdio.h>
#include <string.h>
#include <webots/distance_sensor.h>
#include <webots/emitter.h>
#include <webots/motor.h>
#include <webots/receiver.h>
#include <webots/robot.h>
#include <Windows.h>
#define COMMUNICATION_CHANNEL 1
// time in [ms] of a simulation step
#define TIME_STEP 16
#define PI 3.141592654
#define MAX_SPEED 6.28
typedef enum { EMITTER, RECEIVER } robot_types;
// entry point of the controller
int main(int argc, char *argv[]) {
  // initialize the Webots API
  wb_robot_init();
robot_types robot_type;
int i;
// OK, to copy the values
double x, y, z;
  int message_printed = 0;
      double left_speed  = 0.5 * MAX_SPEED;
    double right_speed = 0.5 * MAX_SPEED;
  // internal variables
 //WbNodeRef robot_node1 = wb_supervisor_node_get_from_def("Emitter E-puck");
 // WbFieldRef trans_field1 = wb_supervisor_node_get_field(robot_node1, "translation");
//  WbNodeRef robot_node2 = wb_supervisor_node_get_from_def("Receiver E-puck");
 // WbFieldRef trans_field2 = wb_supervisor_node_get_field(robot_node2, "translation");
  const double *pos1;
 // const double *pos2;
  //double a[100][];
  WbDeviceTag communication;
    WbDeviceTag gps1 = wb_robot_get_device("gps1");
   wb_gps_enable(gps1, TIME_STEP);
    WbDeviceTag gps2 = wb_robot_get_device("gps2");
 wb_gps_enable(gps2, TIME_STEP);
  WbDeviceTag ps[8];
  char ps_names[8][4] = {
    "ps0", "ps1", "ps2", "ps3",
    "ps4", "ps5", "ps6", "ps7"
  };
  if (strncmp(wb_robot_get_name(), "Emitter E-puck", 14) == 0) {
    robot_type = EMITTER;
    communication = wb_robot_get_device("emitter");
/* if the cannel is not the right one, change it */
    const int channel = wb_emitter_get_channel(communication);
```

```
[gps_zig] MY_ROBOT  hut is at position: -0.259319 0.0485523 0.304041
[gps_zig] MY_ROBOT  hut is at position: -0.258929 0.0485507 0.303124
[gps_zig] MY_ROBOT  hut is at position: -0.258538 0.0485493 0.302206
[gps_zig] MY_ROBOT  hut is at position: -0.258147 0.0485481 0.301287
```

Figure: The Final Design made by compiling all 4 phases - **A** represents the communicated coordinates of one robot to the other which it then displays on the console. **B** represents the controller code. **C** represents the environment wherein as we see two e-Pucks are following the zigzag algo with both Pens set as white in this illustration

Upcoming Milestones

1. IROS Papers Systematic Literature Review.
2. Implementing multi-robot coverage algorithms on WeBots.
3. Shifting to ARGoS for true-swarm-robot behaviour simulation.
4. Applying Reinforcement Learning/Genetic Algorithms to make the robots “learn” by themselves the optimal coverage route when given a known map.
5. Modeling non-convex (non-polygonal) environments as these present further complications to a swarm system, but are more close to real-life environments.

Oxygen Concentration Interfacing

Ashutosh Mishra

PROJECT TITLE:

INTERFACE AN OXYGEN SENSOR TO PIC MICROCONTROLLER TO DISPLAY THE CONCENTRATION OF OXYGEN IN THE LOCAL SURROUNDING. .

SENSOR:

- Sensors are sophisticated devices that are frequently used to detect and respond to electrical or optical signals. A Sensor converts the physical parameter (for example: temperature, blood pressure, humidity, speed, etc.) into a signal which can be measured electrically.

MICROCONTROLLER:

- A microcontroller is a small and low-cost microcomputer, which is designed to perform the specific tasks of embedded systems like displaying microwave’s information, receiving remote signals, etc.
- The general microcontroller consists of the processor, the memory (RAM, ROM, EPROM), Serial ports, peripherals (timers, counters), etc.

Software and development tools used :

- MPLab IDE : Used to develop embedded designs for most microcontrollers and digital signal controllers.
- xC8 compiler
- MPLAB PICkit 3 (To burn the program)