COS 120 – Computing that Matters

Project 2: Pick 1

Spring 2017

For the second two-week project, choose one of the following projects and implement it using the GoPiGo. You should build on what you did in Project 1 and in the exercises. Additional information about the projects is available on the module's webpage.

1 Requirements

- Choose one of the projects below, design, implement, and test it. (You may choose a different project after clearing it with the instructor.)
- You will demonstrate your project in class, and you will submit a short report about the project and a video of your robot in action. Late work will be penalized up to 1/2 a letter grade/day.

2 Project ideas

A. Surveillance bot

Make the Gopigo into a surveillance bot. The robot will either patrol an area and, when it sees something unusual, approach and take a picture of it; or it will lurk where it has a good view of an area, then when it sees something unusual, approach and take a picture.

"Unusual" here could mean something that is unexpected and that you define, such as a purple object in a room that should not have something purple in it, or it could mean something that appears that wasn't there before, such as a different-colored object or a new object detected by sonar. Note that the latter (something new appearing) will require that your robot have an idea of what is *not* new, i.e., a model of its world of some sort.

To do this project, you should build on the behaviors and behavior-based controller you developed in Project 1 and the exercises. You may need to develop new behaviors that make use of the camera, or that create (or at least have) a map of the world.

This project may require you to change the control mechanism from a simple subsumption controller (as in Project 1) to something more tailored to your needs.

Stretch goal: If you want to go further, you could have the robot send email to you when it detects something, including the photo it took in the email. Note that this will involve using Raspbian's mail system, which may require some additional software or configuration; talk to the CLA/instructor if you want to do this.

B. Robopet

For this project, you will turn your Gopigo into a robotic pet. You will build on the behaviors developed in the exercises and Project 1, adding additional ones to imitate pet-like behaviors.

For example, if you were to make a robocat, you might add behaviors to have the robot rub up against your leg, chase prey (e.g., a ball, or the spot of a laser pointer), etc. (Note: just having the robot do nothing, or ignore you, and calling it a robocat, isn't sufficient!) A robodog might do tricks (e.g., based on seeing a particular color via the camera, or the sonar detecting the left-to-right motion of an object, etc.), come when called (when you use a color or wave an object, e.g.), or play fetch.

This project, like project option A., may require you to change the controller from the simple subsumption controller used in Project 1.

Stretch goal: To go further, consider hooking a microphone to the robot to allow it to hear you call it, or to use a computer vision library (e.g., CV) that can do facial recognition to have it recognize its "owner".

C. Mazerunner

For this project, you will have your robot navigate a maze. You will need to make the walls of the maze out of something the sonar can "see" easily. You will build a maze, then have the robot either find a goal in the center or find its way through/out of the maze. You should build on the behavior-based controller from Project 1 and the exercises. You will need to add behaviors, for example, to remember where the robot has been, to choose which direction to go when presented with a choice, etc. You may also need to modify existing behaviors (e.g., so that the robot can approach a wall more closely or not run away, etc.).

You may need to change the controller from the simple subsumption controller used in Project 1.

Stretch goal: To go further, you could create a behavior that maps the maze, so that if you put the robot in it again, it can directly get out.

D. Dribbling

In this project, you will have the Gopigo dribble a soccer ball (or a smaller ball, perhaps) down the "field" and through a "goal". The goal can just be two boxes arranged with a gap between them. The robot will need to be able to locate the ball (e.g., by color or with its sonar), go to it, and move it toward and through the goal, chasing it down as necessary if it gets away.

You can modify the robot so that it can more easily dribble the ball, for example, by taping cardboard or wooden "arms" to it, etc.

You will build in the behavior-based controller from Project 1 and the exercises for this, and you may need to change the controller from the simple subsumption controller.

Stretch goal: To go further, consider dribbling around obstacles on the field, remembering that you may not be able to "see" the obstacles while dribbling, since the ball may block the sonar. You could also consider cooperating with another player—you—to move a ball downfield. In this case, the robot would need to pass the ball to you, receive the ball from you, etc.

E. Robosoccer

This project is a more advanced version of project option D.. Here, you will set up *two* goals and pit your robot against another group's robot in a game of one-on-one soccer.

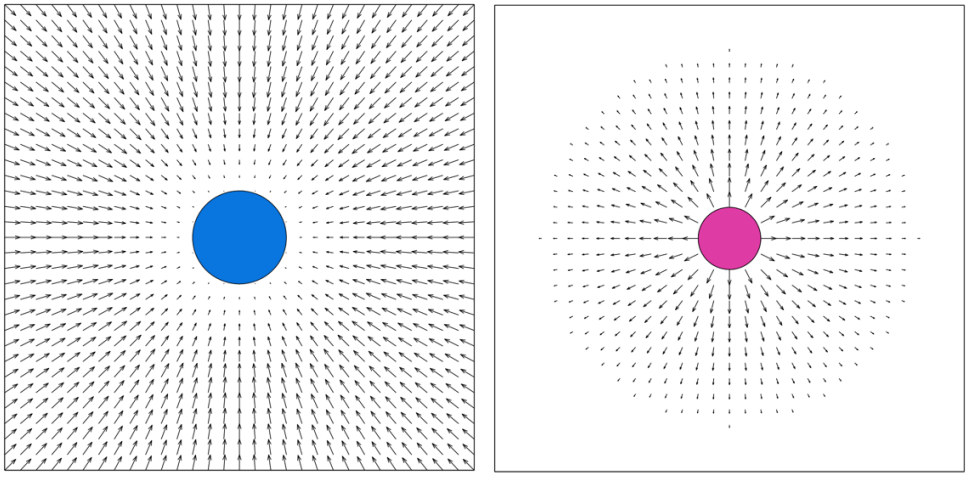
F. Alternative robot controller

In this project, you will explore alternate kinds of robot controllers. At minimum, you will

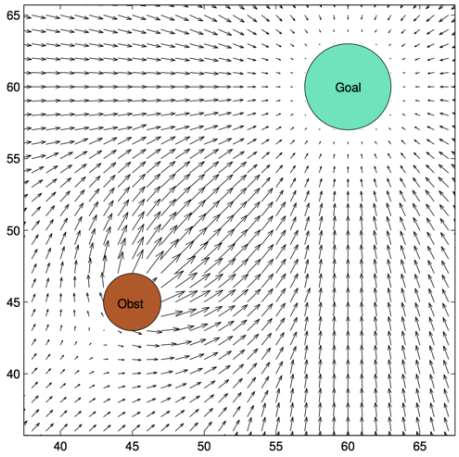
implement a *vector field-based controller* (sometimes called a *potential field-based controller*)[1] and compare it to the behavior-based controller you built in Project 1.

A vector field-based controller models the world as being permeated by a field, much like an electromagnetic field. The value of the field at any location is a vector: a magnitude + a direction. This acts like a force on the robot causing it to go in the direction. In this kind of controller, obstacles are modeled as a source of repulsive force, and goal locations are modeled as source of attractive force.

For example, a goal and an obstacle would generate a potential fields that look like (respectively):[1]

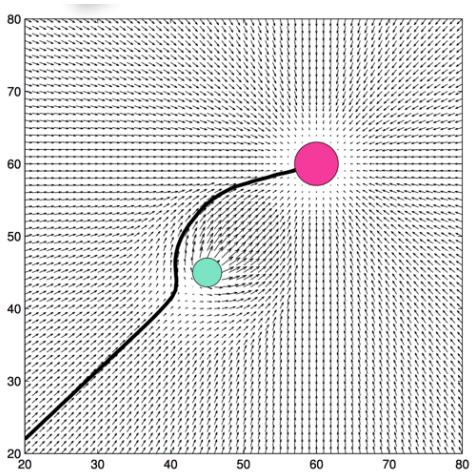


Combining these would give a field something like:



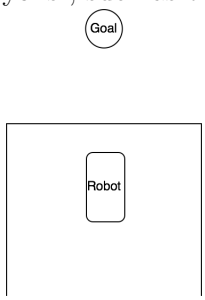
which might lead a robot to follow a path like:

[1]All figures here are from Michael A. Goodrich's class notes entitled *Potential Fields Tutorial*. The tutorial is cited by others, but the definitive citation is difficult to pin down; Dr. Goodrich is now at Brigham Young University. A copy can be found at CiteSeer at `http://citeseerx.ist.psu.edu/viewdoc/download;jsessionid=46D17FA017182E6A1B94820E50ED8F49?doi=10.1.1.115.3259&rep=rep1&type=pdf`.



Note that this is a kind of behavior-based robotics, too, but here, the behaviors are not explicitly programmed, but rather emerge from the potential field's values as the robot moves.

To go further, you might address a problem vector-field robots have with getting stuck in "box canyons", such as this:



Here, the robot is attracted to goal and repelled from the walls, and the vector field won't let it out of the "canyon" that it has gotten into.

3 Evaluation rubric

A	The project was well-designed and implemented and the code was documented and well-written
B	The project was well-designed and implemented with at most minor problems; the code was mostly well-documented and well-written.
C	The project design and implementation may have had some minor problems but few major ones; the code was documented and acceptably-written.
D	The project design and/or implementation had some major flaws; the code was sparsely documented or not documented at all, and may have been poorly-written.
F	The project was incomplete or not attempted.