Hybrid reciprocal velocity obstacle method (HRVO)

1. GOAL

The goal of this practical session is to work with an implementation of a dynamic obstacle avoidance algorithm in multirobot systems. After the session you will be able to define a mutirobot setup with mobile agents and dynamic obstacles to produce free collision navigation of the robots. In particular, this session is based on the implementation of the following paper: Jamie Snape, Jur P. van den Berg, Stephen J. Guy, Dinesh Manocha. The Hybrid Reciprocal Velocity Obstacle. IEEE Trans. Robotics 27(4). 696-706 (2011).

2. METHODOLOGY AND SETUP

Before starting the exercise, read carefully the complete instructions of the practice. The practical session consists of installing the necessary software to make work the obstacle avoidance program. It is also required to modify the code to show the resultant trajectories of the robots and perform optional tasks.

We recommend using the IDE (Integrated Development Environments) of "Visual Studio Code" (VSCode) to edit the code of this session. Nevertheless, any other text editor will do. A list of popular environments can be found here:
http://wiki.ros.org/IDEs

If not installed, check appendix (A) for instructions to install VSCode.

The source code used in this session is in C++ and CMake 3.10 or higher is required to compile the code. Check appendix (B).

3. HYBRID RECIPROCAL VELOCITY OBSTACLE METHOD

The source code of the obstacle avoidance algorithm is available in the repository GitHub with the corresponding documentation:
https://github.com/snape/HRVO
https://www.jamiesnape.io/HRVO/
https://gamma.cs.unc.edu/HRVO

To install this package follow these commands in a terminal:
>> cd
>> git clone https://github.com/snape/HRVO.git
>> cd HRVO
>> cmake .
>> cmake --build .
Execute the compiled example Circle.cpp
>> ./examples/Circle

This program Circle.cpp sets a scenario in which 250 agents, initially positioned evenly distributed on a circle, move to the antipodal position on the circle. In this setup, there are no obstacles apart from the other agents. The code of this example (HRVO/examples/Circle.cpp) is explained in appendix (C).

As a result, the evolution of the agents' coordinates is shown in text in the terminal screen. This is a lot of numbers. Therefore, there is no graphical interface. In order to show graphically the results, there are several options: (a) Program in Circle.cpp the code (C++) to plot the robots along the time in the screen using, for example, OpenCV drawing functions. (b) Save the results in a text file, and then read the results with any other program to show them in the screen (Python, Matlab, etc.). (c) Any other approach you make up.

Some hints for (a) and (b) are given in the appendixes. In particular for (a), appendix (D) summarizes how to install OpenCV in Ubuntu (in case it is not already installed). Then, appendix (E) explains some basics about how to use OpenCV to show the program results. Finally writing the results in a text file (b) for later manipulation in another environment is tackled in appendix (F).

4. OPTIONAL TASKS

Task H1: Define a set of robots distributed in the border of a square and set their goal position in the contour of the first letter of your name, in capitals. Run the program and check the lack of collisions.

Task H2: Define a set of static obstacles in the environment. Run the program and check the lack of collisions. Increase the number of obstacles to analyze the limits of the algorithm.

Task H3: Define a set of dynamic obstacles with predefined trajectories. Run the program and check the lack of collisions. Tune the speed of the dynamic obstacles to analyze the limits of the algorithm.

Task H4: Simulate the results of the collision avoidance algorithm in ROS + Gazebo using a robotic platform (for example: turtlesim, turtlebot, etc.).

5. SESSION REPORT

As a result of this practical session, the final developed code will be submitted through the ADD (https://moodle.unizar.es/add/).

Appendix (A) How to install Visual Studio Code (VSCode)

Go to the official web and download the appropriate version of the program:
https://code.visualstudio.com/Download
If you download the .deb file for Ubuntu, run on a terminal the following command with the file just downloaded:
>> sudo dpkg -i ./code_1.52.1-1608136922_amd64.deb

If it asks for the pip installer, run this command:
>> sudo apt-get install python3-pip
If there is an error with rospy (ImportError: No module named 'yaml') use Python 2.7 instead of Python 3:
>> sudo apt-get install python-pip

Install the Python extension for Visual Studio Code:
https://marketplace.visualstudio.com/items?itemName=ms-python.python
Configuration of Python extension:
https://code.visualstudio.com/docs/python/python-tutorial

Install the C/C++ extension for Visual Studio Code:
https://marketplace.visualstudio.com/items?itemName=ms-vscode.cpptools
Using C++ on Linux in VS Code:
https://code.visualstudio.com/docs/cpp/config-linux

Appendix (B) How to install/update CMake in Ubuntu

Follow these instructions:
(https://cmake.org/cmake/help/latest/guide/tutorial/index.html)
Check the CMake version with:
>> cmake --version
If you are running a version lower than CMake 3.10, you need to install a recent version. Warning: Do not do remove previous version of CMake if you have Robot Operating System (ROS) installed, otherwise removing CMake will also remove parts of your ROS distribution, breaking everything and forcing you to re-install ROS.
Go to the official CMake webpage (http://www.cmake.org/download), then download and extract the latest version. Update the version and build variables in the following commands to get the desired version. In particular:
>> version=3.11
>> build=1
>> mkdir ~/temp
>> cd ~/temp
>> wget https://cmake.org/files/v$version/cmake-$version.$build.tar.gz
>> tar -xzvf cmake-$version.$build.tar.gz
>> cd cmake-$version.$build/

Install the extracted source by running:
>> ./bootstrap

>> make -j$(nproc)
>> sudo make install

Since we kept the previous version of cmake at /usr/bin/cmake, running
>> hash -r cmake
forces the shell to re-examine your PATH and find its current location at /usr/local/bin/cmake

Just test your new cmake version:
>> cmake --version

Appendix (C) Example Circle.cpp

```
/**
 * \file   Circle.cpp
 * \brief  Example with 250 agents navigating through a circular environment.
 */

#ifndef HRVO_OUTPUT_TIME_AND_POSITIONS
#define HRVO_OUTPUT_TIME_AND_POSITIONS 1
#endif

#include <cmath>

#if HRVO_OUTPUT_TIME_AND_POSITIONS
#include <iostream>
#endif

#include <HRVO.h>

using namespace hrvo;

const float HRVO_TWO_PI = 6.283185307179586f;

int main()
{
	Simulator simulator;

	simulator.setTimeStep(0.25f);

	//Default parameters to setup each agent. Details of the parameters in:
//https://www.jamiesnape.io/HRVO/classhrvo_1_1_simulator.html#a54848993c608b7df2cfbf0f99ddbd0c5

	simulator.setAgentDefaults(15.0f, 10, 1.5f, 1.5f, 1.0f, 2.0f);

	//This loop creates 250 agents. Maybe it is too much to start with. Reduce the number to, for example, four. Then, the distribution around the circle should be multiplied for ratio 1/4=0.25 instead of 1/250=0.004. So, change 0.004f with 0.25f
	for (std::size_t i = 0; i < 250; ++i) {
		const Vector2 position = 200.0f * Vector2(std::cos(0.004f * i * HRVO_TWO_PI), std::sin(0.004f * i * HRVO_TWO_PI));
		simulator.addAgent(position, simulator.addGoal(-position));
```

```
	}

	do {
#if HRVO_OUTPUT_TIME_AND_POSITIONS
		//Prints in the terminal the time of each iteration:
		std::cout << simulator.getGlobalTime();
		//Prints in the terminal the coordinates (x, y) of each robot of each iteration:
		for (std::size_t i = 0; i < simulator.getNumAgents(); ++i) {
			std::cout << " " << simulator.getAgentPosition(i);
		}

		std::cout << std::endl;
#endif /* HRVO_OUTPUT_TIME_AND_POSITIONS */

		simulator.doStep();
	}
	while (!simulator.haveReachedGoals());

	return 0;
}
```

Appendix (D) How to install OpenCV in Ubuntu

Follow the instructions in this web:
https://docs.opencv.org/master/d7/d9f/tutorial_linux_install.html
In particular:
Install minimal prerequisites
>> sudo apt update && sudo apt install -y cmake g++ wget unzip
Download and unpack sources
>> wget -O opencv.zip https://github.com/opencv/opencv/archive/master.zip
>> unzip opencv.zip
Create build directory
>> mkdir -p build && cd build
Configure
>> cmake ../opencv-master
Build (this takes some time)
>> cmake --build .

Appendix (E) How to use OpenCV to show the program results

See the examples in this source code to draw shapes with OpenCV:
https://docs.opencv.org/3.4/d3/d96/tutorial_basic_geometric_drawing.html
https://sodocumentation.net/opencv/topic/9749/drawing-shapes--line--circle-------etc--in-cplusplus

In order to compile the code with the OpenCV functions you inserted in Circle.cpp, you need to link OpenCV libraries in your program. General instructions are provided here:
https://docs.opencv.org/master/db/df5/tutorial_linux_gcc_cmake.html
In particular, you need to modify the file: /HRVO/examples/CMakeLists.txt
Add the following lines at the beginning of the file:

```
find_package( OpenCV REQUIRED )
include_directories( ${OpenCV_INCLUDE_DIRS} )
add_executable(Circle Circle.cpp)
```

And then, add the term ${OpenCV_LIBS} in line 50 (approximately) such that:

```
 target_compile_definitions(Circle PRIVATE
   ${HRVO_EXAMPLES_COMPILE_DEFINITIONS})
 target_link_libraries(Circle PRIVATE ${HRVO_LIBRARY} ${OpenCV_LIBS})
```

Appendix (F) How to write a text file using C++

Some useful code to write the results in a text file:

```
#include <fstream>

std::ofstream outfile;
outfile.open("result.txt"); // open a file in write mode.

outfile << simulator.getGlobalTime(); // writes time in the file
outfile << " " << simulator.getAgentPosition(i); // writes agent's i position

outfile << std::endl; // writes an end of line
```