RefineCBF Code Documentation

Introduction

Documentation to be completed soon!

Cloning the Package

The package is constructed in such a way that it is advised to initially create its own workspace directory before checkout.

```
mkdir ~/refinecbf_ws
```

Now in the root of your workspace, clone the code:

```
cd ~/refine_cbf_ws && git clone https://github.com/ncussonn/turtwig.git
```

Building the Package

Navigate to the root of your workspace where the package is and build the package:

```
cd ~/refinecbf_ws && colcon build —symlink-install
```

Package Anatomy

- **src:** source code folder, contains all ros packages to be built
    - **refine_cbf**:

    - **launch:**
        - `refine_cbf_launch.py`
    - **refine_cbf:**
        - `config.py`
        - `dynamic_programming.py`
        - `experiment_obstacles.py`
        - `high_level_controller.py`
        - `low_level_controller.py`
        - `manual_controller.py`
        - `nominal_policy.py`
        - `refine_cbf_visualization.py`
        - `safety_filter.py`
        - `tf_stamped_2_odom.py`
        - `utils.py`
    - `package.xml`
    - `setup.py`
- **DynamixelSDK**: standard Turtlebot3 package that runs the Dynamixel motors
- **turtlebot3**: standard Turtlebot3 package with other generic Turtlebot3 packages contained within
- **turtlebot3_msgs**: Turtlebot3 library containing custom messages
- **turtlebot3_simulations**: *augmented* simulation package containing example Gazebo world with custom obstacle models
    - **turtlebot3_gazebo:**
        - **launch:**
            - `refine_cbf_experiment_2x2.launch.py`
            - `empty_world.launch.py`

- `precomputed_cbf.npy`
- `nominal_policy_table.npy`
- `refine_cbf.rviz`

General Comments

The package is built around modified ROBOTIS Turtlebot3 standard libraries and Dynamixel SDK libraries. As such, if other Turtlebot3 (TB3) packages are sourced at the same time in your environment, there may be erroneous behavior.

Safety Filter Node

Description

This node solves a CBF-QP optimization problem to find the minimally invasive safe-control action and publishes to

Running the Node:

`cd ~/refinecbf_ws && ros2 run refine_cbf safety_filter`

Nominal Policy Node

`cd ~/refinecbf_ws && ros2 run refine_cbf nominal_policy`

Dynamic Programming Node

`cd ~/refinecbf_ws && ros2 run refine_cbf dynamic_programming`

RefineCBF Visualization Node

`cd ~/refinecbf_ws && ros2 run refine_cbf refine_cbf_visualization`

Running an Example

An example has been provided in the package which provides a Gazebo environment and associated nominal policy table and initial CBF. Upon installation of the package, the example can be immediately run by doing the following:

1. Open 3 terminals, we will call these terminals A, B, and C.
2. Source foxy in all 3 terminals.
3. In A, run rviz2 and load the refine_cbf rviz2 file
4. In B, source the refine_cbf package and launch the gazebo example world
5. In C, source the refine_cbf package and launch the RefineCBF network

Upon completing these steps, the robot in Gazebo should begin moving and in RVIZ2 you should see various contours representing obstacles, the current CBF, initial CBF, goal, and turtlebot trajectory (as seen in the Figure below).

Figure

The robot will attempt to reach the goal region while remaining in the current safe set. The square obstacles should disappear at particular iterations of the algorithm. Eventually, the safe set will encompass the goal and the TB3 can safely navigate and orbit in / about the goal.

The user can play with all parameters in the config file besides the goal for this example to get a feel for how to customize the implementation to their own needs.