RoboCupJunior Rescue Simulation 2023

Технический журнал команды

Таёжные Ёжики

Аннотация:
команда Таёжные Ёжики представляет вам описание алгоритмов и робота для участия в RoboCupJunior Rescue Simulation 2023

Состав нашей команды:

- Пильщиков Григорий Андреевич - программировал, сделал тренировочный лабиринт.
- Цыганкова Мария Сергеевна - программировала, разбиралась с работой датчиков в webots, в том числе и с лидаром.
- Косаченко Сергей Викторович - тренер команды

Наши победы:

- 3 место на Робофинисте в регламенте Robocup junior rescue maze.
- 1 место на Кубке губернатора в регламенте соревнования роботов с техническим зрением именем Ширшина.

Цель:

- Цель команды заключается в создании алгоритма, для обследования комнат и построения карты, а также для обнаружения жертв и предупредительных знаков опасности

План:

- Сначала команда разработала тренировочное поле и тренировочного робота в симуляторе Webots, где начала осваивать азы работы в данной среде. После написания первого алгоритма перемещения в лабиринте команда создала собственного робота в формате json, и стала тестировать его в Erebus. Где смогла начать обучаться с работай других датчиков (лидар камера и т. д.), после тренировок, а также многочисленных доработок команда усовершенствовала алгоритм работы робота.

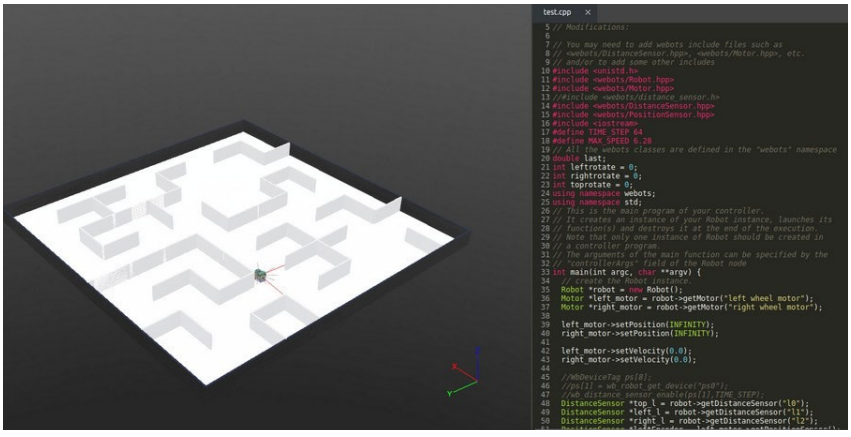
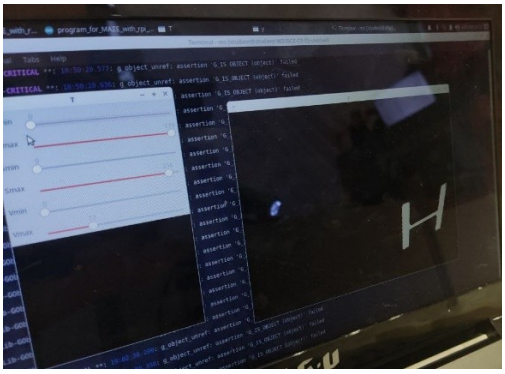
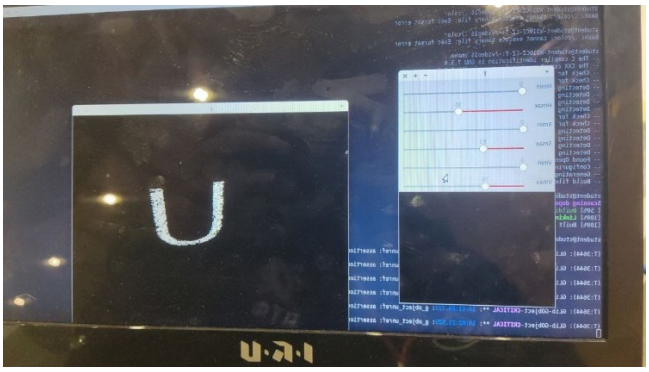


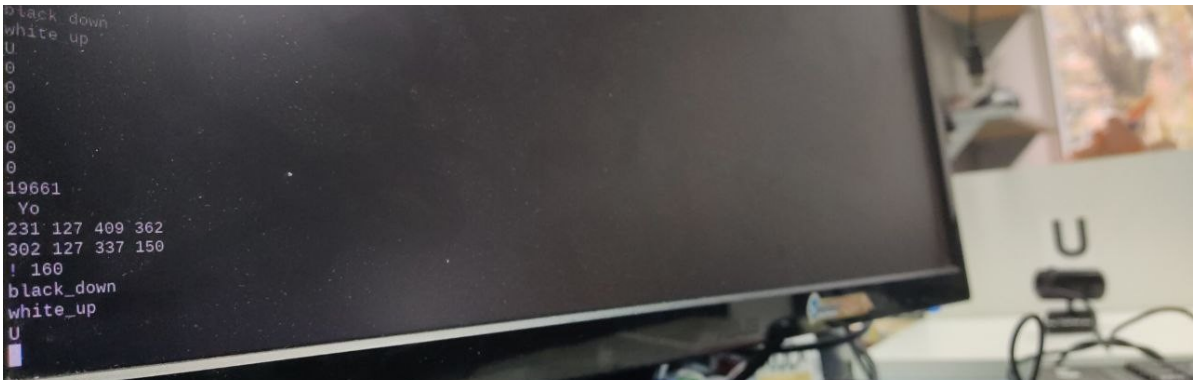
Фото тренировочного поля и робота с

- Во время работы мы столкнулись с несколькими проблемами, например, проблемой ограниченности частей робота (мы не можем прикрепить дополнительное освещение или большое количество датчиков).

- Кроме того основным циклом симулятора является цикл while (robot->step(timeStep) != -1) из-за чего накладываются ограничения на использование других циклов while так как симулятор требует вернуться к началу timeStep за один внутренний тик. Для решения данной проблемы нам пришлось скорректировать алгоритм без использования while.
- Еще одной проблемой с которой мы столкнулись были большие тени в симуляторе и отсутствие дополнительного освещения на роботе так как для определения типов жертв мы используем библиотеку OpenCV на камере должно быть хорошее освещение, тогда мы заметили, что ColorSensor подсвечивает пространство вокруг себя, и поставили его в качестве фонарика.
- Этой осенью наша команда ездила на соревнования в Санкт-Петербург, во время подготовки к ним был разработан алгоритм распознавания жертв для Raspberry Pi Zero, благодаря чему у нашей команды был алгоритм, при корректировке которого робот смог обнаруживать жертвы в лабиринте.

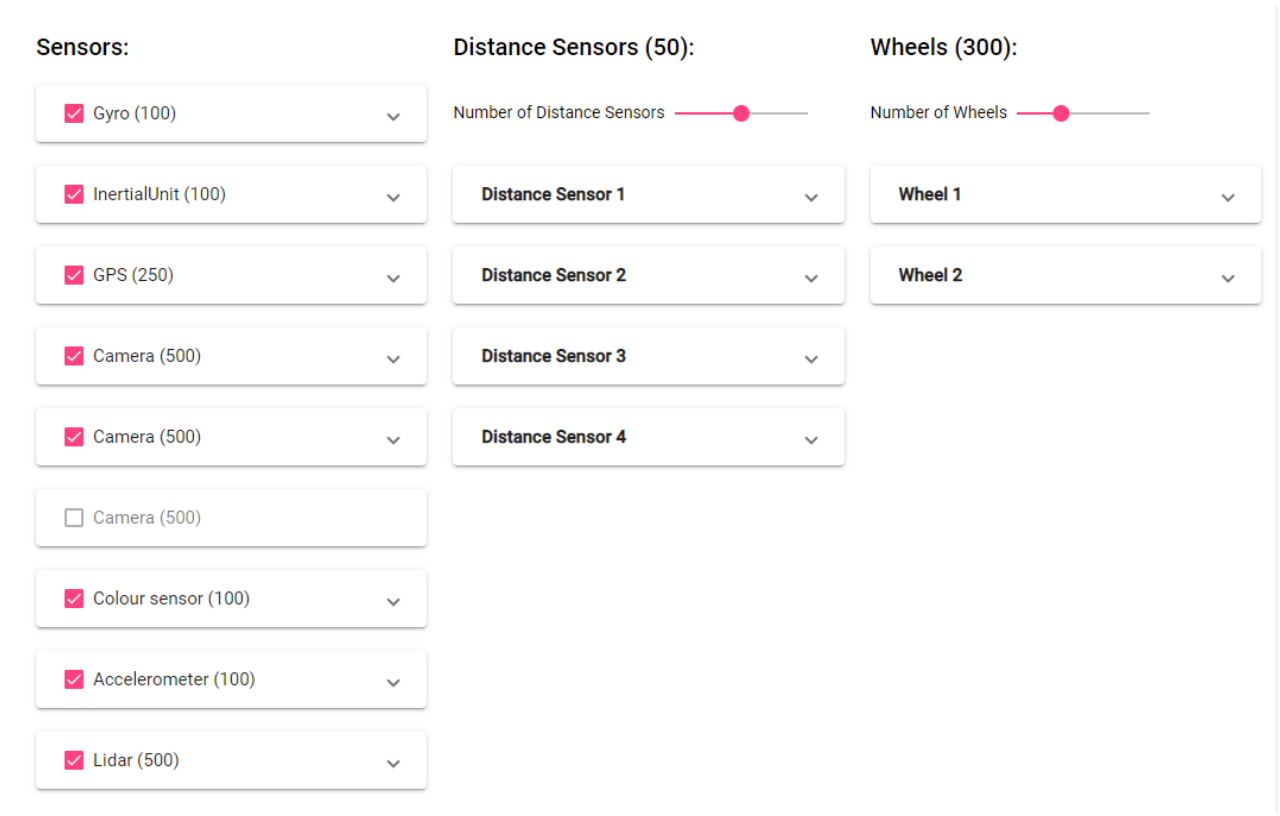


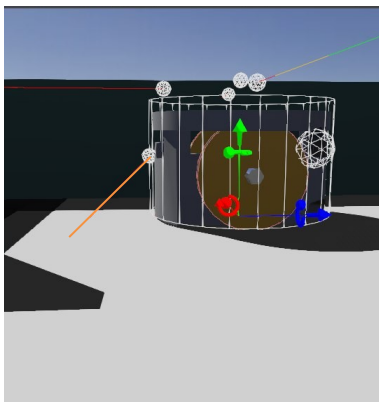




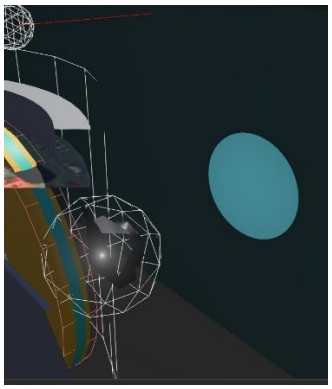
Распознавание жертв в реальном мире

- Наш робот обладает множеством различных датчиков, помогающих эффективно обследовать карту (подробнее можно увидеть на рисунке).





Хочется отметить наличие DistanceSensor, который смотрит в пол, что позволяет на ходу предугадывать наличие ямы спереди.



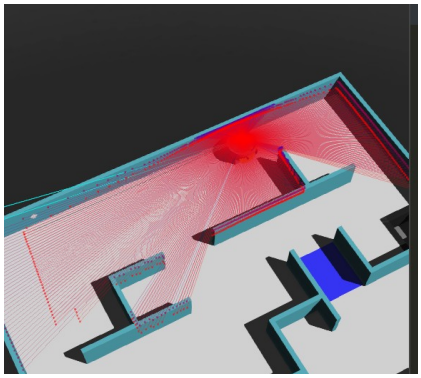
Для решения проблемы с освещенностью камеры, был добавлен ColorSensor, используемый нами как фонарик.

В самом начале нашей программы мы объявляем библиотеки и переменные, дальше расписываются структуры и функции для распознавания жертв и Hazmat signs. Далее начинается main в котором до цикла объявляются и инициализируются всякие возможные датчики и моторы, в бесконечном цикле, привязанном к timeStep (while (robot->step(timeStep) != -1)), вначале идёт проверка переехал ли робот в новый тайл, если переехал, то вносятся начальные показания для некоторых переменных, далее идёт проверка на отсутствие прогресса, если такое обнаруживается, то робот проезжает вперёд - назад для проверки своей ориентации в пространстве посредством компаса и гироскопа. После идёт блок, предназначенный для передвижения робота и напоследок, проверка на антизалипание, что бы если робот застрял, он отъезжал назад...

- Проводя испытания, мы столкнулись с проблемой повторной идентификации жертв, эту проблему нам удалось исправить посредством привязки жертвы к таилу на котором она располагается, т.е. если мы повторно посетим этот таил мы увидим, что жертва уже обнаружена.

Кроме того, была выявлена проблема зацикливания одного маршрута, нам удалось решить эту проблему придумав «правила смены рук».

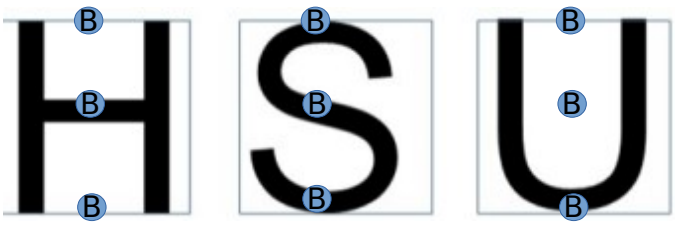
- Алгоритм навигации робота построен на карте, изначально, робот высчитывает координаты gps переднего и левого таила, отдавая предпочтения поворота на неисследованную территорию, если же все доступные пути уже исследованы робот двигается по алгоритму действующей руки.



Наличие стен робот анализирует при помощи лидара

- Для более эффективного передвижения по лабиринту было разработано «правило смены рук». При множественном посещении таила робот будет сменять ведущую руку в зависимости четности/нечетности количества посещений клетки.

- Жертвы и Hazmat signs мы обнаруживаем с помощью датчика камеры и библиотеки OpenCV 4. Для начала настраивается бинаризация для 1 из 3 цветов: красного, жёлтого и чёрного. После бинаризации определяем контуры и проверяем самые большие контуры, подходят ли они. Далее если нашлись подходящие контуры для жёлтой бинаризации, то мы определили Hazmat sign- Organic Peroxide, если же не найдены подходящие жёлтые, но есть подходящие красные, то это Flammable Gas, если же обнаружены чёрные контуры, подходящего размера, то вычисляем 3 точки прямоугольника этого контура: верхнюю срединную, среднюю срединную и нижнюю срединную, с помощью них мы и определяем какая буква и ищем в них контур на чёрной бинаризации: если верхний, средний и нижний - чёрные, то это s, если нижний и верхний - не чёрные, а средний - чёрный, то это H, а если нижний - чёрный, а средний и верхний - нет, то это U;



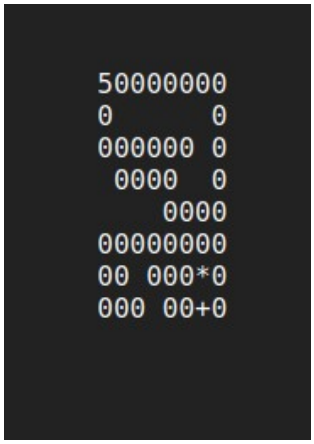
Точки, анализируемые нами

- Изучая OpenCV мы сильно расширили, свои знания в компьютерном зрении.

[Explain the testing procedures you used to validate your code and present relevant testing data]

- Для испытания алгоритма нахождения жертв, мы выключали роботу моторы, подносили к необходимому символу и анализировали правильность показаний.

- Для построения карты наша команда решила использовать двумерный массив из структур, в каждую из которой при посещении мы вносим данные об наличии жертвы в ней, если жертва есть, то о её типе, о типе пола (старт/чекпоинт/обычный и т.д.), о наличии стенок вокруг тайла и о количестве прохождений этого самого тайла. На этих данных и строиться алгоритм поведения нашего робота в лабиринте. На данном этапе мы не имеем никаких slam и a* алгоритмов. Наш робот, просто на основе посещений тайлов выбирает по какой руке ехать: по правой или по левой. А с помощью запоминания ячеек, в которых находятся жертвы наш робот не распознаёт их дважды.



- В процессе работы над системой картостроения мы познакомились с структурами и работой с ними, что помогло нам улучшить свои умения в программировании.
- Для проверки своего кода мы запускали нашего робота в лабиринт и, отслеживая его исследованные тайлы, следили за правильностью строимой роботом картой.

Построенная карта для дальнейшего передвижения по полю

- На данный момент, разработанный нами робот может перемещаться по карте занося данные в небольшую карту, которую позже может использовать для навигации.
- Все испытания робота проходили непосредственно в Erebus, однако в некоторых случаях гораздо эффективнее испытывать робота предварительно отключив ему моторы.
- Было замечено множество недостатков алгоритма, в частности алгоритма построения маршрутов, поэтому в будущем планируется разработать более сложные алгоритмы, для более эффективного обследования карты.

- Изучая принципы работы того или иного алгоритма, придумывая более оптимальные системы получения результата команда продвинулась в программировании.

Ссылки на используемые ресурсы

- https://www.youtube.com/@StormingRobots - Storming Robots;
- https://www.youtube.com/@KajalGada - Kajal Gada;
- https://www.youtube.com/@roboticknowledge Robotic Knowledge.

Благодарности

Наша команда благодарит нашего тренера - Косаченко Сергея Викторовича, а так же ОГБОУ «Томский Физико - Технический Лицей» за предоставленные ноутбуки

Ссылки на нас

Наш Github - https://github.com/Grin2020/TE2022;