Командный практический тур

В командной части заключительного этапа участники должны спроектировать автономную систему управления групой роботов-погрузчиков для решения задач навигации на модели логистического центра с использованием алгоритмов компьютерного зрения.

Командная часть заключительного этапа проходит в течении 3,5 дней (всего 26 астрономических часов), которые включают работу по оснащению роботов необходимыми датчиками, программированию, пробные заезды на макете логистического центра, зачетные попытки.

Легенда

Недалёкое будущее, в автоматических логистических центрах практически нет людей, всю работу выполняют роботы-погрузчики, которыми управляет интеллектуальное ПО по распределению задач.

Несмотря на отсутствие человеческого фактора, не следует думать, что в таких центрах никогда не будет проблем. Форс-мажор может произойти в любой момент и система из роботов и ПО должна уметь справляться с такими ситуациями.

Поэтому для финальной задачи профиля «Интеллектуальные робототехнические системы» предлагается рассмотреть следующий эпизод: в логистическом центре произошла перезагрузка всех систем, что привело к сбросу информации о местоположениях работающих в данный момент роботов-погрузчиков, а также карты данной части логистического центра.

В начале выполнения задания считается, что робототехнические устройства активируются в каких-то секторах (каждый в своём) логистического центра. Один из роботов выполняет считанные с arTag маркера команды, а затем ожидает прибытия второго робота в смежный сектор. Первый робот должен найти второго, доехав до смежного с ним сектора, а после вернуться в один из секторов старта. Структура логистического центра не известна заранее. При перемещении роботы не должны

сталкиваться, а также повреждать логистический центр.

Задача участников Олимпиады — разработать программу управления несколькими робототехническими устройствами для выполнения задания описанного выше.

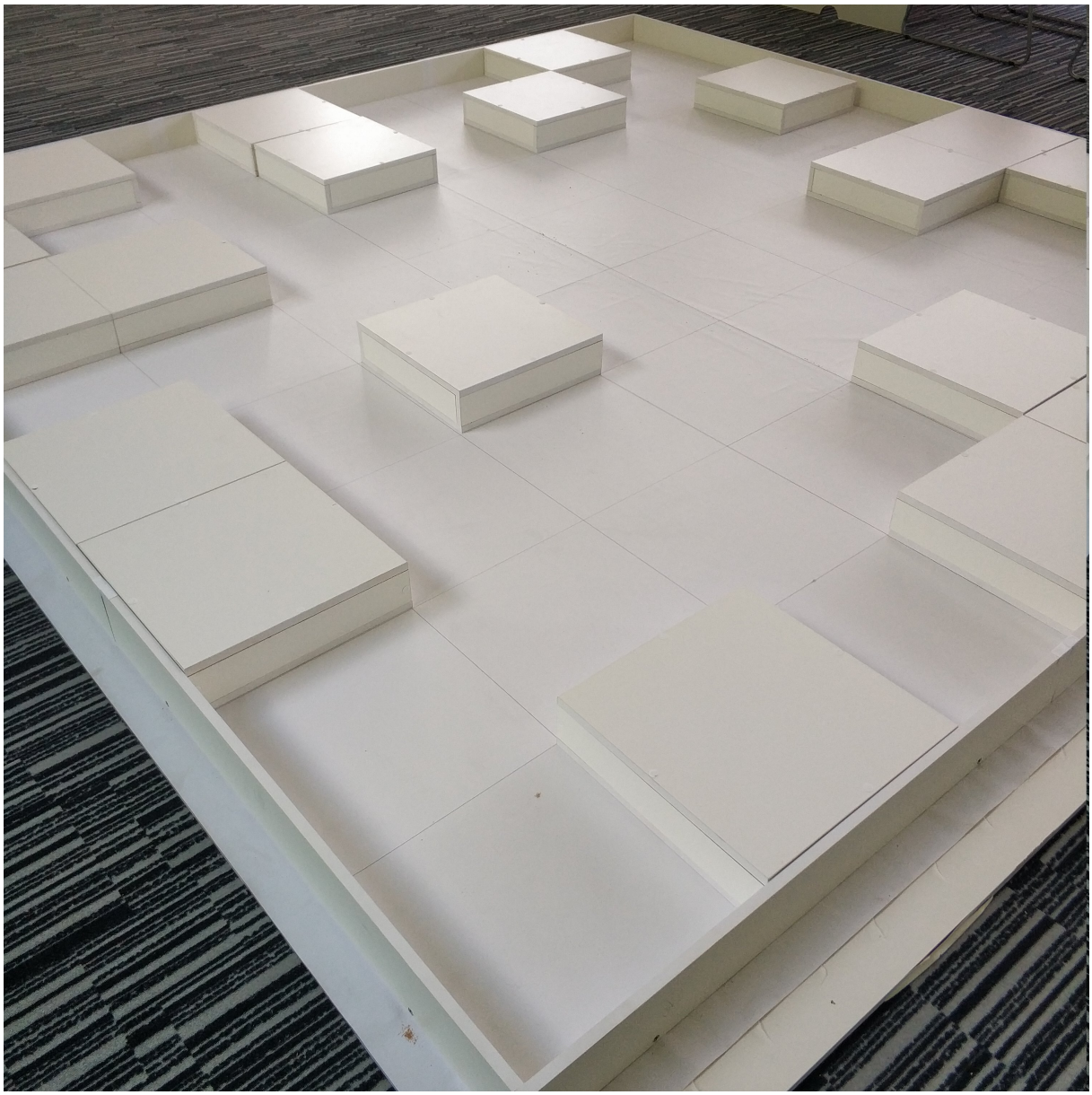

Рис. III.2.1: Полигон для запуска робототехнических устройств на финале Олимпиады НТИ

Набор заданий

Решение командной задачи разбито на четыре этапа. Первые три этапа итеративно подводят участников к решению полной финальной задачи, осуществляемому во время последнего четвёртого этапа. На каждом этапе в проверку решения заданий данного этапа входят:

- способность проверить гипотезу о работоспособности алгоритма через демонстрацию решения в симуляторе;

- полнота решения задания конкретного этапа;
- воспроизводимость результатов — робототехническое устройство участников должно неоднократно выполнить требуемые действия.

Первый этап

Задача: робототехническое устройство располагается в модели логистического центра. Ему необходимо проехать по маршруту в точку окончания работы. Структура логистического центра заранее неизвестна.

Включая содержательные задачи:

- Нахождение порогового значения для датчиков расстояния;
- Реализация алгоритмов перемещения в неизвестной среде.
- Реализация алгоритмов сегментации действий.

Второй этап

Задача: два робототехнических устройства должны определить своё местоположение в модели логистического центра (структура которого заранее неизвестна) из случайных, заранее неизвестных точек старта.

Включая содержательные задачи:

- Реализация коммуникации между робототехническими устройствами;
- Реализация алгоритмов локализации для группы робототехнических устройств;
- Реализация алгоритмов перемещения в неизвестной среде.
- Реализация алгоритмов движения нескольких роботов в замкнутой среде.

Третий этап

Задача: робототехническое устройство должно проехать по модели логистического центра заданные команды. Команды робот должен определить самостоятельно по ARTag метке, расположенной на стеллаже, прилегающему к сектору старта.

Включая содержательные задачи:

- Калибровка камеры робототехнического устройства;
- Реализация алгоритмов компьютерного зрения: считывание ARTag меток без использования дополнительных библиотек;
- Декодирование бинарного кода с использованием кода Хэмминга.

Четвёртый этап

Задача: два роботехнических устройства располагаются в неизвестной среде. Первый робот должен обнаружить второго робота и после вернуться в одну из начальных позиций. Второй робот должен выполнить заданные команды, переданные через ArTag маркер, расположенный в секторе старта.

Включая содержательные задачи:

- Реализация коммуникации между робототехническими устройствами;
- Реализация алгоритмов составления карты неизвестной местности и локализация на данной местности;
- Реализация алгоритмов компьютерного зрения: считывание ARTag меток без использования дополнительных библиотек;
- Декодирование бинарного кода, с использованием кода Хэмминга;
- Реализация алгоритмов движения нескольких роботов в замкнутой среде.

Описание модели логистического центра

Полигон - квадратное поле 3200 × 3200 мм., разделенное на квадратные сектора 400 × 400 мм. Некоторые сектора недоступны для посещения робототехническим устройством и представляют из себя модель стеллажа высотой 100 мм.

Полигон окружен бортом высотой 100 мм. Конфигурация полигона изменяется и определяется непосредственно перед каждым запуском роботов.

На стеллаже, прилегающего одной из четырех сторон к сектору активации (старта), на высоте от 100-150 мм от уровня поверхности поля закреплен ARTag маркер (https://goo.gl/WaTFMB), определяющий серию команд, предназначенную для выполнения роботом. Размер маркера - 40 × 40 мм. Маркер обращен лицевой стороной внутрь сектора старта данного робота. Конкретная высота расположения маркеров определяется в первый день финала и остается постоянной на все дни финального этапа. При этом допустимая погрешность установки маркеров ±5 мм. Пример расположения маркера на стелаже представлен на рисунке III.2.2

Рис. III.2.2: Стеллаж с установленным ARTag маркером

Маркер состоит из 8 × 8 элементов одинакового размера. Элементы маркера, расположенные по его границе — всегда черные. Четыре элемента, находящиеся в углах внутреннего 6 × 6 квадрата определяют ориентацию маркера таким образом, что только один из них — белый. Оставшиеся 32 элемента маркера кодируют число

по следующему правилу: если элемент черный, то он обозначает 1, если белый, то 0 при этом самый первый элемент — старший бит закодированного числа. Нумерация элементов относительно ориентационных элементов обозначена на рисунке III.2.3.

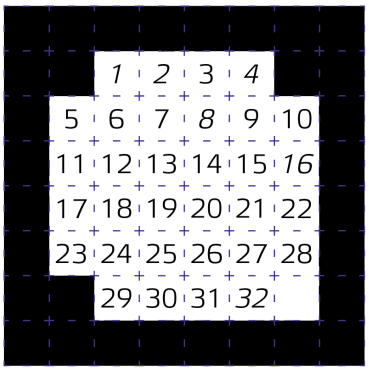

Рис. III.2.3: Нумерация элементов маркера относительно ориентационных элементов

Закодированное на маркере двоичное число использует код Хэмминга (https://habr.com/ru/post/140611/), в котором 26 информационных битов и 6 контрольных:

1 Первый контрольный бит;
2 Второй контрольный бит;
3 Старший бит первой команды $K_1 (0 \leqslant K_1 \leqslant 3)$;
4 Третий контрольный бит;
5 Младший бит первой команды $K_1 (0 \leqslant K_1 \leqslant 3)$;
6 Старший бит второй команды $K_2 (0 \leqslant K_2 \leqslant 3)$;
7 Младший бит второй команды $K_2 (0 \leqslant K_2 \leqslant 3)$;
8 Четвёртый контрольный бит;
9 Старший бит третьей команды $K_3 (0 \leqslant K_3 \leqslant 3)$;
10 Младший бит третьей команды $K_3 (0 \leqslant K_3 \leqslant 3)$;
11 Старший бит четвёртой команды $K_4 (0 \leqslant K_4 \leqslant 3)$;
12 Младший бит четвёртой команды $K_4 (0 \leqslant K_4 \leqslant 3)$;
13 Старший бит пятой команды $K_5 (0 \leqslant K_5 \leqslant 3)$;
14 Младший бит пятой команды $K_5 (0 \leqslant K_5 \leqslant 3)$;
15 Старший бит шестой команды $K_6 (0 \leqslant K_6 \leqslant 3)$;
16 Пятый контрольный бит;
17 Младший бит шестой команды $K_6 (0 \leqslant K_6 \leqslant 3)$;
18 Старший бит седьмой команды $K_7 (0 \leqslant K_7 \leqslant 3)$;
19 Младший бит седьмой команды $K_7 (0 \leqslant K_7 \leqslant 3)$;
20 Старший бит восьмой команды $K_8 (0 \leqslant K_8 \leqslant 3)$;
21 Младший бит восьмой команды $K_8 (0 \leqslant K_8 \leqslant 3)$;
22 Старший бит девятой команды $K_9 (0 \leqslant K_9 \leqslant 3)$;
23 Младший бит девятой команды $K_9 (0 \leqslant K_9 \leqslant 3)$;
24 Старший бит десятой команды $K_{10} (0 \leqslant K_{10} \leqslant 3)$;
25 Младший бит десятой команды $K_{10} (0 \leqslant K_{10} \leqslant 3)$;
26 Старший бит одиннадцатой команды $K_{11} (0 \leqslant K_{11} \leqslant 3)$;
27 Младший бит одиннадцатой команды $K_{11} (0 \leqslant K_{11} \leqslant 3)$;

28 Старший бит двенадцатой команды $K_{12}(0 \leqslant K_{12} \leqslant 3)$;
29 Младший бит двенадцатой команды $K_{12}(0 \leqslant K_{12} \leqslant 3)$;
30 Старший бит тринадцатой команды $K_{13}(0 \leqslant K_{13} \leqslant 3)$;
31 Младший бит тринадцатой команды $K_{13}(0 \leqslant K_{13} \leqslant 3)$;
32 Шестой (последний) контрольный бит;

Данные команды задают действия, которые должен выполнить робот-погрузчик:

0 Действие не требуется (“N”);
1 Роботу необходимо повернуть налево на 90° внутри данного сектора (“L”);
2 Роботу необходимо повернуть направо на 90° внутри данного сектора (“R”);
3 Роботу необходимо проехать в следующий по направлению движения сектор (“F”);

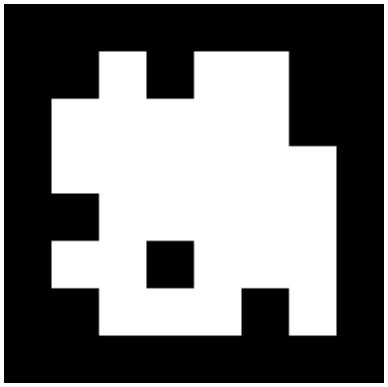

Рис. III.2.4: Маркер с закодированным значением - $10111111101111110111111101111110_2$

Маркер на рисунке III.2.4 кодирует число $10111111101111110111111101111110_2$ — $K_1 = 3$, $K_2 = 3$, $K_3 = 2$, $K_4 = 3$, $K_5 = 3$, $K_6 = 2$, $K_7 = 3$, $K_8 = 3$, $K_9 = 3$, $K_{10} = 2$, $K_{11} = 3$, $K_{12} = 3$, $K_{13} = 3$.

Гарантируется, что робот начинает свое движение в секторе, к сторонам которого прилегает как минимум один стеллаж.

Сектора активаций роботов (стартов) никак не обозначаются на поле и определяются непосредственно перед каждым заездом робота.

В некоторых задачах используются координаты, в таком случае, сектор $(0, 0)$ находится в левом верхнем углу, ось X направлена вправо, ось Y – вниз.

Описание конструктора

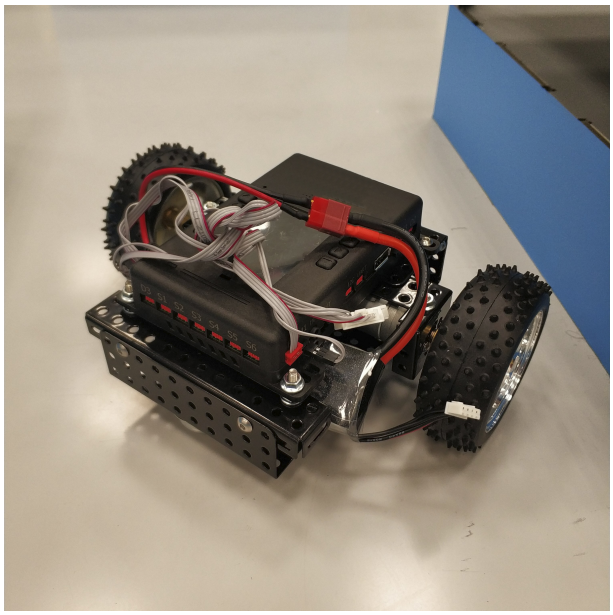

Рис. III.2.5: Мобильная платформа TRIK в сборе

В первый день финального тура каждой команде выдаются два комплекта:

- Инструкция по сборке наземной платформы на базе конструктора TRIK. Мобильная платформа построена по принципу дифференциального управления. Физические размеры платформы позволяют совершать все маневры внутри одного сектора модели логистического центра без касания со стенками стеллажей или бортов.
- Комплект деталей для сборки мобильной наземной платформы на базе конструктора TRIK (блок управления TRIK, аккумулятор, два мотора с энкодерами на датчиках Холла, колеса).
- Комплект дополнительных деталей из конструктора TRIK и набор датчиков: 2 инфракрасных датчика дальности, 2 ультразвуковых датчика расстояния и 1 VGA-камера;
- комплект дополнительных деталей из конструктора TRIK;
- Ноутбуки с установленной TRIK Studio, каждой команде не более двух ноутбуков. При этом участники могут пользоваться своими ноутбуками.

Условия проведения

1. Из полученного набора датчиков команды могут выбирать те, с помощью которых, по мнению участников, можно решить задачу наиболее эффективным способом.
2. Команды могут вносить любые изменения в мобильные наземные платформы.
3. Участники во время командного этапа финального тура могут использовать интернет и заранее подготовленные библиотеки для решения задачи.

4. Участники не могут использовать помощь тренера, сопровождающего лица или привлекать третьих лиц для решения задачи.
5. Финальная задача формулируется участникам в первый день финального тура, но участники выполняют решение задачи поэтапно. Критерии прохождения каждого этапа формулируются для каждого дня финального тура. За подзадачи, решенные в конкретном этапе начисляются баллы. Баллы за подзадачи можно получить только в день, закреплённый за конкретным этапом.
6. Некоторые подзадачи строго требуют выполнение каких-то предыдущих подзадач. Выполнение данных подзадач без выполнения предыдущих допускается, однако данная попытка будет оценена в 0 баллов.
7. Во время рабочего времени команды могут проводить испытания на полигоне. Количество подходов, которое может сделать команда может быть ограничено в зависимости ограничений, накладываемых расписанием финального этапа Олимпиады.
8. Испытания на полигоне должны осуществляться так, чтобы не мешать другим командам, проводящим в это время свои испытания на полигоне. Для этого всем командам может быть назначено ограничение по времени, которое они могут тратить на одно испытание. После истечения этого времени, команда должна дать возможность проводить испытания следующей команде.
9. Часть подзадач необходимо будет решить в симуляторе TRIK Studio: команда получает 3 тестовых виртуальных полигона с соответствующими наборами входные данных для подготовки решения, в то время как приемка решения происходит на расширенном наборе полигонов для проверки универсальности управляющей программы. Начисление баллов за подзадачи может происходить только в тот этап, в котором данные подзадачи сформулированы.
10. У команды есть неограниченное количество попыток решения подзадач в симуляторе:
    10.1. Решения принимаются на проверку до истечения первых 6,5 часов работы в соответствующий соревновательный день. Время может меняться в зависимости от дня.
    10.2. До истечения 6,5 часов, команда должна загрузить свое решение на gitlab.com в соотвествующий репозиторий дня, внутри своей группы, доступ к которой участники получат в начале олимпиады.
    10.3. До истечения указанного времени команды могут изменять файл с решением сколько угодно раз. Проверяться будет всегда только последняя доступная версия.
    10.4. Зафиксировать последнее решение необходимо, создав Merge Request (MR).
    10.5. При создании Merge Request в качестве ответственного (assignee) укажите того, кто будет отвечать за приемку результатов.
11. Часть подзадач для реальных роботов может быть запрещена к приемке без успешного прохождения 60% всех тестов, предназначенных для проверки решения соответствующей подзадачи в симуляторе.
12. Каждый день финального тура за 2 часа (может варьироваться в зависимости от расписания) до конца выделенного рабочего времени команды должны сдать роботов в зону карантина. Время сдачи роботов в карантин может изменяться и зависит от количества команд и сложности подзадач, принимаемых в конкретный этап.
13. Перед сдачей робота в карантин команды должны загрузить на роботов управ-

ляющие программы, подготовленные для демонстрации решения задачи, а также ее копию в репозиторий, доступ к которому участники получат в начале соревновательного дня. Без программы, загруженной в репозиторий, команды не допускаются до проверки решения на реальном роботе.

14. Зафиксировать загрузку файлов, содержащих решения задачи на реальном роботе, необходимо при помощи MR.
15. Программа должна успешно собираться в pdf.
16. После момента, когда все роботы сданы в карантин, судьи по одной вызывают команды для приемки решения подзадач, закрепленных за этапом конкретного дня финального тура.
17. Может быть предусмотрено до двух попыток сдачи решения одной и той же подзадачи на реальных роботах. Конкретное количество попыток определяется в конкретных подзадачах.
18. После прохождения приемочных запусков, баллы набранные командой заносятся судьями в протокол. Один из участников команды расписывается за набранный результат, подтверждая согласие команды с оценкой проведенных запусков.
19. Роботы должны выполнять задание полностью автономно. Удаленное управление не допускается. Касание какого-либо робота участником команды после его старта во время приемочных запусков не допускается. Алгоритм, реализующий систему управления группой роботов, должен планировать свое выполнение, полагаясь только на информацию с датчиков.
20. Введение данных в программу до старта устройства (например, координат робота в начале работы) разрешается только для тех задач, где это явно прописано. Во всех других случаях введение данных в программу роботов перед запуском запрещено.
21. Для всех роботов программа должна быть одинаковой, допускается отличие лишь в разрешенных входных данных.
22. Если какая-то подзадача подразумевает считывание информации с элементов, расположенных на полигоне, запрещается при запуске роботов вводить информацию о положении этих элементов или значениях, которые данные элементы определяют.
23. Если во время приемочных запусков у судьи возникли сомнения о том, что задачи подэтапа решены корректно (роботы не выполняют задачу полностью автономно, участник вводит значения в каких-либо роботов перед запуском), то он вправе провести инспекцию кода. По результатам инспекции, судья вправе снять с команды баллы, набранные за данный этап.
24. Если во время приемочных запусков у судьи возникает ситуация, когда он не может однозначно решить выполняются ли критерии решения подзадачи, он вправе принять решение не в пользу команды.
25. В случае если возникает техническая проблема, независящая от участников, то по решению судей может быть предоставлена возможность перезапуска
26. Команда вправе обсуждать с судьей результаты приемочных запусков до вызова следующей команды, но финальное решение остается о начислении баллов остается за судьей.

Процедура проведения приемочных запусков и критерии оценки

Первый этап

1. Командам необходимо подготовить две задачи для симулятора:
   1.1. В качестве первой задачи участникам необходимо выполнить следующее:
      1.1.1. Робот устанавливается в модели логистического центра в заранее неизвестном секторе. Задача робота выполнить все команды переданные через входной файл. Робот выполнил все команды и вывел `finish` на всех проверочных полигонах.
      Входные данные: через файл `input.txt` управляющей программе передаются:
         A. В первой строке через пробел — команды которые необходимо выполнить, где:
            0 Действие не требуется (“N”);
            1 Роботу необходимо повернуть налево на 90° внутри данного сектора (“L”);
            2 Роботу необходимо повернуть направо на 90° внутри данного сектора (“R”);
            3 Роботу необходимо проехать в следующий по направлению движения сектор (“F”);

      Ожидаемый результат: После запуска программы робот выполнил все команды, остановился и вывел на экран `finish`.
   1.2. В качестве второй задачи:
      1.2.1. Робот устанавливается в модели логистического центра в заранее неизвестном секторе. Задача робота проехать из точки старта в точку финиша. Робот проехал из точки старта в точку финиша и вывел `finish` на всех проверочных полигонах. Координаты сектора старта, включая направление и финиша передаются через входной файл. Структура лабиринта не известна.
      Входные данные: через файл `input.txt` управляющей программе передаются:
         A. В первой строке через пробел — координаты старта X_s, Y_s, направление старта D_s ($0 \leqslant X_s, Y_s \leqslant 7$, $D_sin[0, 3]$):
            0 Робот направлен вверх (в сторону отрицательного направления оси Y);
            1 Робот направлен направо (в сторону положительного направления оси X);
            2 Робот направлен вниз (в сторону положительного направления оси Y);
            3 Робот направлен налево (в сторону отрицательного направления оси X);
         B. Во второй строке через пробел — координаты финиша X_f, Y_f, $0 \leqslant X_f, Y_f \leqslant 7$;

      Ожидаемый результат: После запуска программы робот доехал до финиша, остановился и вывел на экран `finish`.

1.3. Правила именования файлов с управляющей программой для проверки решений в симуляторе:
   1.3.1. Для первой задачи: `sim_1.js`;
   1.3.2. Для второй задачи: `sim_2.js`.

2. Конфигурация робота в симуляторе следующая:
   2.1. Конфигурация моторов:
      2.1.1. M3 — правый мотор;
      2.1.2. M4 — левый мотор.
   2.2. Конфигурация датчиков:
      2.2.1. D1 — ультразвуковой датчик, направленный вперёд;
      2.2.2. D2 — ультразвуковой датчик, направленный назад;
      2.2.3. A1 — инфракрасный датчик, направленный вправо;
      2.2.4. A2 — инфракрасный датчик, направленный влево.
3. Команде необходимо будет подготовить решения для двух разных подзадач для реальных роботов. На демонстрацию каждого решения предоставляется 2 попытки.
4. Все попытки осуществляются 18 марта.
5. За 5 мин до сдачи в карантин для 1ой подзадачи судья определяет количество и набор команд для выполнения роботом.
6. После сдачи в карантин для 1ой подзадачи судья определяет сектор старта и направление робота в секторе старта.
7. За 5 мин до сдачи в карантин для 2ой подзадачи судья определяет сектор старта, направление старта и сектор финиша для робота. Данные значения команда должна внести в программу перед тем, как сдать робота в карантин.
8. Секторы старта могут быть различными в разных попытках. Данные значения команда должна внести в программу перед тем, как сдать робота в карантин.
9. Правила именования файлов с программой управления:
   9.1. для первой попытки первой подзадачи: `real_1_1.js`;
   9.2. для второй попытки первой подзадачи: `real_1_2.js`;
   9.3. для первой попытки второй подзадачи: `real_2_1.js`;
   9.4. для второй попытки второй подзадачи: `real_2_2.js`.
10. Merge Request необходимо назвать следующим образом: “код команды_day1”. Например: “irs202000_day1”.
11. Максимальное время выполнения одной попытки для 1ой подзадачи — 2 минуты.
12. Максимальное время выполнения одной попытки для 2ой подзадачи — 5 минут.
13. Баллы за решение задач этапа:
   13.1. **Первая задача в симуляторе**: робот смог выполнить всю задачу на всех проверочных полигонах — 12 баллов.
   13.2. **Вторая задача в симуляторе**: робот смог выполнить всю задачу на всех проверочных полигонах — 20 баллов.
   13.3. **Первая подзадача на реальном роботе:** Робот устанавливается в модели логистического центра в заранее неизвестном секторе. Задача робота выполнить все команды. Робот выполнил все команды и вывел `finish`. — 16 баллов.
   13.4. **Вторая подзадача на реальном роботе:** Робот устанавливается в мо-

дели логистического центра в заранее неизвестном секторе. Задача робота проехать из точки старта в точку финиша. Робот проехал из точки старта в точку финиша и вывел `finish`. Координаты сектора старта, включая направление и финиша передаются через входной файл. Структура лабиринта не известна. — 20 баллов.

14. Выводить `finish` следует не менее 10 секунд.
15. Баллы за первую подзадачу не начисляются, если не было частично засчитано решение первой задачи в симуляторе
16. Баллы за вторую подзадачу не начисляются, если не было частично засчитано решение второй задачи в симуляторе.
17. Баллы за все попытки в каждой подзадаче суммируются.
18. Выполнение всех критериев в каждой из двух попыток всех двух подзадач дает дополнительные 8 баллов.
19. Максимальное количество баллов за этап — 112.

Второй этап

1. Командам необходимо подготовить две задачи для симулятора:
   1.1. В качестве первой задачи участникам необходимо выполнить следующее:
      1.1.1. Робот устанавливается в модели логистического центра в заранее неизвестном секторе. Необходимо в процессе движения опредлить своё местоположение, остановиться и вывести на экран координаты робота в формате "(X, Y)" без пробелов и без кавычек. Направление старта передаётся через входной файл.
      Входные данные: через файл `input.txt` управляющей программе передаются:
         A. В первой строке — направление старта D_s:
            0 Робот направлен вверх (в сторону отрицательного направления оси Y);
            1 Робот направлен направо (в сторону положительного направления оси X);
            2 Робот направлен вниз (в сторону положительного направления оси Y);
            3 Робот направлен налево (в сторону отрицательного направления оси X);
      Ожидаемый результат: После запуска программы робот определил своё местоположение, остановился и вывел на экран `finish`.
   1.2. В качестве второй задачи:
      1.2.1. Робот устанавливается в модели логистического центра в заранее неизвестном секторе. Известно что один сектор данного логистического центра заблокирован другим роботом. Координаты этого сектора передаются через входной файл. Необходимо в процессе движения определить своё местоположение и приехать в сектор, смежный с сектором нахождения другого робота и вывести `finish`. Структура лабиринта заранее не известна.
      Входные данные: через файл `input.txt` управляющей программе передаются:
         A. В первой строке через пробел — направление старта D_s

0 Робот направлен вверх (в сторону отрицательного направления оси Y);

1 Робот направлен направо (в сторону положительного направления оси X);

2 Робот направлен вниз (в сторону положительного направления оси Y);

3 Робот направлен налево (в сторону отрицательного направления оси X);

B. Во второй строке через пробел — координаты расположения другого робота X, Y ($0 \leqslant X_s, Y_s \leqslant 7$);

Ожидаемый результат: После запуска программы робот доехал до одного из смежных секторов с сектором нахождения другого робота, остановился и вывел на экран `finish`.

1.3. Правила именования файлов с управляющей программой для проверки решений в симуляторе:

1.3.1. Для первой задачи: `sim1.js`;

1.3.2. Для второй задачи: `sim2.js`.

2. Конфигурация робота в симуляторе следующая:

2.1. Конфигурация моторов:

2.1.1. M3 — правый мотор;

2.1.2. M4 — левый мотор.

2.2. Конфигурация датчиков:

2.2.1. D1 — ультразвуковой датчик, направленный вперёд;

2.2.2. D2 — ультразвуковой датчик, направленный назад;

2.2.3. A1 — инфракрасный датчик, направленный вправо;

2.2.4. A2 — инфракрасный датчик, направленный влево.

3. Команде необходимо будет подготовить решения для одной задачи для реальных роботов. На демонстрацию решения предоставляется 2 попытки.

4. Все попытки осуществляются 19 марта.

5. Merge Request необходимо назвать следующим образом: “код команды_day2”. Например: “irs202000_day2”.

6. За 10 мин до сдачи в карантин судья определяет направление старта для робота. Данное значение команда должна внести в программу перед тем, как сдать робота в карантин.

7. Направление старта могут быть различными в разных попытках. Данные значения команда должна внести в программу перед тем, как сдать робота в карантин.

8. После сдачи в карантин судья определяет секторы старта обоих роботов.

9. Правила именования файлов с программой управления:

9.1. для первой попытки задачи для реального робота: `real_1_1.js`;

9.2. для второй попытки задачи для реального робота: `real_1_2.js`;

10. Максимальное время выполнения одной попытки - 7 минут.

11. Баллы за решение задач этапа:

11.1. **Первая задача в симуляторе**: робот смог определить своё местоположение на всех проверочных полигонах — 16 баллов.

11.2. **Вторая задача в симуляторе**: робот смог достичь смежного сектора

на всех проверочных полигонах — 20 баллов.

11.3. **Задача на реальном роботе:** Роботы располагаются в случайных секторах робототехнического полигона.

11.3.1. Первый робот определил своё местоположение на поле, остановился, издал звуковой сигнал, вывел на экран свои координаты в формате (X, Y). — 14 баллов.

11.3.2. Координаты второго робота также определены. Второй робот издал звуковой сигнал, вывел на экран свои координаты в формате (X, Y). — 10 баллов.

11.3.3. Во время решения посталенной задачи второй робот не покидал изначальных сектор. — 8 баллов.

12. Выводить значение метки следует не менее 10 секунд.
13. Баллы за задачу на реальном роботе не начисляются, если не было частично засчитано решение первой задачи в симуляторе.
14. Баллы за все попытки на реальном роботе суммируются.
15. Выполнение всех критериев в каждой из двух попыток всех двух подзадач дает дополнительные 8 баллов.
16. Максимальное количество баллов за этап — 108.

Третий этап

1. В качестве задачи для симулятора участникам необходимо выполнить следующее:

1.1. Робот устанавливается в модели логистического центра в заранее неизвестном секторе. При этом структура логистического центра не известна заранее. Задача робота проехать из точки старта в точку финиша, согласно маршруту, заданному изображением ARTag маркера, заранее считанным с камеры реального устройства. На финише необходимо вывести на экран слово `finish`.

Входные данные: через файл `input.txt` управляющей программе передаются:

- В первой строке 19200, разделенных пробелом, целых чисел $P_{1,i}$ $(0 \leqslant P_{1,i} \leqslant 2^{24})$ — изображение ARTag маркера;

Каждое число в маркере — точка, закодированная в формате RGB, т.е. строка с изображением маркера эквивалентна снимку разрешением 160×120 точек.

Ожидаемый результат: После запуска программы робот доехал до финиша, остановился и вывел на экран `finish`.

1.2. Имя файла с управляющей программой для проверки решения в симуляторе: `sim1.js`.

2. Конфигурация робота в симуляторе следующая:

2.1. Конфигурация моторов:

2.1.1. M3 — правый мотор;

2.1.2. M4 — левый мотор.

2.2. Конфигурация датчиков:

2.2.1. D1 — ультразвуковой датчик, направленный вперёд;

2.2.2. D2 — ультразвуковой датчик, направленный назад;

2.2.3. A1 — инфракрасный датчик, направленный вправо;

2.2.4. A2 — инфракрасный датчик, направленный влево.

3. Команде необходимо будет подготовить решения для двух разных подзадач для реального робота. На демонстрацию каждого решения предоставляется 2 попытки.
4. Все попытки осуществляются 20 марта.
5. Merge Request необходимо назвать следующим образом: “код команды_day3”. Например: “irs202000_day3”.
6. Допускается возможность демонстрации первой подзадачи во время периода отладки.
7. Сектор старта может быть разным для каждой попытки.
8. Правила именования файлов с программой управления:
   8.1. для первой подзадачи: `real_1.js`;
   8.2. для второй подзадачи: `real_2.js`;
9. Максимальное время выполнения одной попытки - 3 минуты.
10. Баллы за решение задач этапа:
    10.1. **Задача в симуляторе**: робот проехал из точки старта в точку финиша на всех проверочных полигонах — 16 баллов.
    10.2. **Первая подзадача на реальном роботе:** Робот распознал верно ARTag метку, которая установлена прямо перед камерой, и вывел на экран верные команды, закодированные в ARTag метке. Допускается ручная настройка (наведение) камеры на маркер (включая визуальную корректировку средствами просмотра). — 8 баллов.
    10.3. **Вторая подзадача на реальном роботе:** Робот располагается в секторе старта, с неизвестной стороны которого располагается ARTag метка. Робот нашел метку, издал звуковой сигнал и вывел на экран верные команды, закодированные в ARTag метке — 12 баллов.
11. Цифры, означающие команды, необходимо выводить на экран через пробел. Допускается вывод в несколько строк.
12. За вторую подзадачу начисляется только половина возможных баллов, если не было засчитано решение в симуляторе, а также если не сдана первая подзадача.
13. Выводить значение метки следует не менее 10 секунд.
14. Баллы за все попытки в каждой подзадаче суммируются.
15. Выполнение всех критериев в каждой из двух попыток всех двух подзадач дает дополнительные 4 балла.
16. Максимальное количество баллов за этап — 60.

Четвёртый этап

1. В качестве задачи для симулятора участникам необходимо выполнить следующее:
   1.1. Робот устанавливается в модели логистического центра в заранее неизвестном секторе. Известно что один сектор данного логистического центра заблокирован другим роботом. Необходимо в процессе движения опредлить своё местоположение, доехать в сектор старта и вывести на экран координаты одного из достижимых смежных секторов заблокиро-

ванного робота в формате "(X, Y)" без пробелов и без кавычек.
Входные данные: через файл `input.txt` управляющей программе передаются:

1.1.1. В первой строке — направление старта D_s:
- 0 Робот направлен вверх (в сторону отрицательного направления оси Y);
- 1 Робот направлен направо (в сторону положительного направления оси X);
- 2 Робот направлен вниз (в сторону положительного направления оси Y);
- 3 Робот направлен налево (в сторону отрицательного направления оси X);

1.1.2. Во второй строке через пробел — координаты расположения второго робота X, Y ($0 \leqslant X, Y \leqslant 7$);

Ожидаемый результат: После запуска программы робот определил своё местоположение, вернулся в сектор старта и вывел на экран координаты.

1.2. Имя файла с управляющей программой для проверки решения в симуляторе: `sim1.js`.

2. Конфигурация робота в симуляторе следующая:
   2.1. Конфигурация моторов:
      2.1.1. M3 — правый мотор;
      2.1.2. M4 — левый мотор.
   2.2. Конфигурация датчиков:
      2.2.1. D1 — ультразвуковой датчик, направленный вперёд;
      2.2.2. D2 — ультразвуковой датчик, направленный назад;
      2.2.3. A1 — инфракрасный датчик, направленный вправо;
      2.2.4. A2 — инфракрасный датчик, направленный влево.
3. Команде необходимо будет подготовить решения для одной подзадачи для реальных роботов. На демонстрацию решения предоставляется 1 попытка.
4. Все попытки осуществляются 21 марта.
5. Merge Request необходимо назвать следующим образом: "код команды_day4". Например: "irs202000_day4".
6. Стартовым направлением для первого робота является направление 0.
7. После сдачи в карантин для подзадачи судья определяет секторы старта и направление старта 2го робота.
8. Правила именования файлов с программой управления:
   8.1. для первой попытки подзадачи: `real_1_1.js`;
9. Максимальное время выполнения одной попытки - 15 минут.
10. Баллы за решение задач этапа:
    10.1. **Задача в симуляторе**: робот смог выполнить всю задачу на всех проверочных полигонах — 20 баллов.
    10.2. **Задача на реальном роботе**: Роботы располагаются в секторах старта, задача определить свое местоположение, выполнить вторым роботом команды (относительно начального направления робота), переданные через ARTag метку, расположенную на случайном стеллаже вокруг сектора старта, доехать первым роботом до второго и после до любого сектора

старта.

10.2.1. Второй робот успешно считал ARTag метку, выполнил команды, остановился, издал звук и вывел на экран *finish* и более не двигался. — 24 балла.

10.2.2. Первый робот локализовался, доехал до смежного сектора, издал звук. — 24 балла.

10.2.3. Первый робот доехал до смежного сектора, после перемещения второго робота — 8 баллов.

10.2.4. Первый робот локализовался, доехал до смежного сектора, издал звук. Через 10 сек продолжил движение до любого сектора старта, остановился, издал звук и вывел на экран *finish*.—24 балла.

10.2.5. Первый робот доехал до ближайшей (с точки зрения количества секторов) зоны старта. Перед этим, первый робот доехал до смежного сектора после перемещения второго робота — 16 баллов.

11. Баллы за задачу на реальном роботе не начисляются, если не было частично засчитано решение задачи в симуляторе.
12. Частичным решением в симуляторе считается успешное прохождение 30% всех тестов.
13. Во всемя решения задачи возможен перезапуск. При перезапуске команда может поправить своё решение, потратив на это `не более трёх минут`.
14. При перезапуске набранные баллы не пропадают, время не останавливается.
15. В случае если команда использовала перезапуск, то она получит половину из достигнутых баллов за всё попытку.
16. Баллы за все попытки в каждой подзадаче суммируются.
17. Выполнение всех критериев(не обязательно на максимум баллов) в каждой из двух попыток задачи на реальном роботе дает дополнительные 4 балла.
18. Максимальное количество баллов за этап — 120.

Решение

solution/helpers.ts

```
import { Direction } from './FieldGraph';

export function normalizeAngle(angle: number) {
    var newAngle = angle % 360;
    while (newAngle <= -180) newAngle += 360;
    while (newAngle > 180) newAngle -= 360;
    return newAngle;
}

export function normalizeDirection(direction: number): Direction {
    direction = direction % 4;
    if (direction < 0) direction += 4;
    return direction;
}

export function copy<T>(x: T): T {
    return JSON.parse(JSON.stringify(x));
```

```
}

export function run_generator(generator: Iterable<unknown>): void {
    for (const _ of generator) {
    }
}
```

solution/sensor_wrap.ts

```
import trik_brick from "trik_brick";

export abstract class SensorWrap {
    abstract update(total_elapsed_time: number, delta_time: number): void;
}

export abstract class ScalarSensor extends SensorWrap {
    abstract get_value(): number;
}

export class SmoothUltrasonicSensorWrap extends ScalarSensor {
    sensor: trik_brick.Sensor
    previous_stable_reading: number;
    previous_unstable_reading: number;
    current_reading: number;

    constructor(sensor: trik_brick.Sensor) {
        super();
        this.sensor = sensor;
        this.current_reading = sensor.read();
        this.previous_stable_reading = this.current_reading;
        this.previous_unstable_reading = this.current_reading;
    }

    update(total_elapsed_time: number, delta_time: number): void {
        const currentRead = this.sensor.read();
        if (Math.abs(currentRead - this.previous_unstable_reading) > 10)
        {
            console.log("unstable: " + this.previous_stable_reading + ", " +
            ↪  currentRead);
            this.previous_unstable_reading = currentRead;
            this.current_reading = this.previous_stable_reading;
        }

        this.previous_unstable_reading = currentRead;
        this.previous_stable_reading = this.previous_unstable_reading;

        this.current_reading = currentRead;
    }


    get_value(): number {
        return this.current_reading;
    }
}
```

solution/artag/hamming.ts

```
import { ARTAG_SIZE } from "./libmorat-ng";

const message_len = (ARTAG_SIZE - /*borders*/ 2) ** 2 - /*corners*/ 4;
const control_bits = Math.floor(Math.log2(message_len));
const hamming_masks: number[] = [];

// note: using little-endian for representing binaries

for (let control_bit = 0; control_bit < control_bits; ++control_bit) {
    const index = 2 ** control_bit;
    let bitmask = 0;
    for (let i = index - 1; i < message_len; i += index * 2) {
        for (let di = 0; di < index; ++di) {
            bitmask |= 1 << (i + di);
        }
    }

    hamming_masks.push(bitmask);
}

export function extract_message(hamming: number) {
    let ret = 0;
    let c = 0;
    for (let i = 2; i < message_len; ++i) {
        const l2 = Math.log2(i + 1);
        if (Math.floor(l2) !== l2) {
            ret |= ((hamming >> i) & 1) << c;
            c++;
        }
    }
    return ret;
}

function extend_message(message: number) {
    let ret = 0;
    let c = 0;
    for (let i = 0; i < message_len; ++i) {
        const l2 = Math.log2(i + 1);
        if (Math.floor(l2) !== l2) {
            const mi = message & 1;
            message >>= 1;
            ret |= mi << c;
        }
        c++;
    }
    return ret;
}

function calculate_control_bits(extended_message: number) {
    let ret = extended_message;
    for (
        let control_bit_number = 0;
        control_bit_number < control_bits;
        ++control_bit_number
    ) {
        const index = 2 ** control_bit_number - 1;
        let v = hamming_masks[control_bit_number] & extended_message;
        let bit = 0;
```

```
        while (v > 0) {
            bit ^= v & 1;
            v >>= 1;
        }
        ret |= bit << index;
    }

    return ret;
}

export function encode(message: number) {
    return calculate_control_bits(extend_message(message));
}

export function decode(hamming: number) {
    const recomputed = encode(extract_message(hamming));
    let diff = recomputed ^ hamming;
    let c = 0;
    const different_bits: number[] = [];
    while (diff > 0) {
        c++;
        const bit = diff & 1;
        diff >>= 1;
        if (bit) different_bits.push(c);
    }

    if (different_bits.length > 0) {
        console.log("Found a encode mistake, fixing...")
        const to_correct =
            different_bits.reduce((acc, curr) => acc + curr, 0) - 1;
        return extract_message(hamming ^ (1 << to_correct));
    }
    return extract_message(hamming);
}
```

solution/artag/artag_ decode.ts

```
import { ARTAG_SIZE } from './libmorat-ng';
import { Command } from '../Command';
import { decode } from './hamming';

function decode_to_number(bin_artag: NdArray) {
    let ret = 0;
    let c = 0;
    for (let i = 1; i < ARTAG_SIZE - 1; ++i) {
        for (let j = 1; j < ARTAG_SIZE - 1; ++j) {
            if (
                (i === 1 && j === 1) ||
                (i === 1 && j === ARTAG_SIZE - 2) ||
                (i === ARTAG_SIZE - 2 && j === 1) ||
                (i === ARTAG_SIZE - 2 && j === ARTAG_SIZE - 2)
            ) {
                continue;
            }
            ret |= bin_artag.get(i, j) << c;
            c++;
        }
    }
```

```
    return ret;
}

export function artag_decode(bin_artag: NdArray): Command[] {
    let num = decode_to_number(bin_artag);
    num = decode(num);
    const commands = Array(13)
        .fill(0)
        .map(
            (_, i) => (((num >> (i * 2)) & 1) << 1) | ((num >> (i * 2 + 1)) & 1)
        );
    return commands as Command[];
}
```

solution/artag/libmorat-ng.ts

```
import ndarray from "ndarray";

export const ARTAG_SIZE = 8;

const SHAPE: [number, number] = [120, 160];
const LIGHT_NORM: [number, number] = [0.8741, -16.94];
const DARK_NORM: [number, number] = [1, 0];
const XDARK_NORM: [number, number] = [1.582, 0];

const MAP: [number, number][] = new Array(SHAPE[1]);
MAP.fill(LIGHT_NORM);
MAP[3] = XDARK_NORM;
for (let i = 5; i < SHAPE[1]; i+= 2)
    MAP[i] = DARK_NORM;
MAP[7] = XDARK_NORM;

type pos = [number, number];

export class InvalidArtagImageError extends Error {
}

function filter(image: NdArray) {
    for (let j = 0; j < image.shape[1]; j++) {
        const [slope, intercept] = MAP[j];
        for (let i = 0; i < image.shape[0]; i++) {
            let v = slope * image.get(i, j) + intercept;
            v = Math.max(0, v);
            v = Math.min(255, v);
            image.set(i, j, v);
        }
    }
    for (let j = 0; j < image.shape[1]; j += 2) {
        for (let i = 0; i < image.shape[0]; i++) {
            let v = image.get(i, j) + image.get(i, j + 1);
            v /= 2;
            image.set(i, j, v | 0);
            image.set(i, j + 1, v | 0);
        }
    }
}

function histogram(image: NdArray): number[] {
    const res: number[] = new Array(256);
```

```
    res.fill(0);
    for (let i = 0; i < image.shape[0]; i++) {
        for (let j = 0; j < image.shape[1]; j++)  {
            const v = image.get(i, j);
            res[v]++;
        }
    }
    return res;
}

function threshhold(image: NdArray, thresh: number) {
    for (let i = 0; i < image.shape[0]; i++) {
        for (let j = 0; j < image.shape[1]; j++)  {
            const v = image.get(i, j);
            const nv = (v < thresh) ? 0 : 255;
            image.set(i, j, nv);
        }
    }
}

function otsu(image: NdArray) {
    /* http://www.labbookpages.co.uk/software/imgProc/otsuThreshold.html */
    const total = image.shape[0] * image.shape[1];
    const hist = histogram(image);

    const hist_sum = hist.reduce((a, x, i) => i * x + a, 0);
    let wB = 0, wF = 0, sumB = 0;
    let best_thresh = 0;
    let best_thresh_variance = 0;
    for (let i = 0; i < 256; i++) {
        wB += hist[i];
        if (wB == 0) continue;

        wF = total - wB;
        if (wF == 0) break;

        sumB += i * hist[i];

        const mB = sumB / wB;
        const mF = (hist_sum - sumB) / wF;
        const variance = wB * wF * (mB - mF) * (mB - mF);
        if (variance > best_thresh_variance) {
            best_thresh = i;
            best_thresh_variance = variance;
        }
    }
    threshhold(image, best_thresh);
}

function get_ccs(image: NdArray): pos[][] {
    function* nei(p: pos): Iterable<[number, number]> {
        const [i, j] = p;
        if (i > 0)                  yield [i - 1, j];
        if (i < image.shape[0] - 1) yield [i + 1, j];
        if (j > 0)                  yield [i, j - 1];
        if (j < image.shape[1] - 1) yield [i, j + 1];
    }
    const seen = ndarray(new Array(image.shape[0] * image.shape[1]), image.shape);
    seen.data.fill(0);
    const ccs: pos[][] = [];
```

```
    function bfs(p: pos, cc: number) {
        if (seen.get(...p) !== 0)
            return;
        seen.set(...p, 1);
        const queue: pos[] = [ p ];
        while (queue.length > 0) {
            const p = queue.shift()!;
            ccs[cc].push(p);
            function put(i: number, j: number) {
                if (image.get(i, j) === 0 && seen.get(i, j) === 0) {
                    seen.set(i, j, 1);
                    queue.push([i,j]);
                }
            }
            //for (let x of nei(p)) {
            const [i, j] = p;
            if (i > 0)                  put(i - 1, j);
            if (i < image.shape[0] - 1) put(i + 1, j);
            if (j > 0)                  put(i, j - 1);
            if (j < image.shape[1] - 1) put(i, j + 1);
            //}
        }
    }
    let cc = 0;
    for (let i = 0; i < image.shape[0]; i++) {
        for (let j = 0; j < image.shape[1]; j++)  {
            const p: pos = [i, j];
            if (image.get(i, j) < 128 && seen.get(i, j) == 0) {
                console.log("bfs(" + p + ")");
                ccs.push([]);
                bfs(p, cc);
                cc++;
            }
            //console.log("mining...");
        }
    }
    return ccs;
}

function slice(image: NdArray, from_i: number = 0, to_i: number = image.shape[0],
↪   from_j: number = 0, to_j: number = image.shape[1]): NdArray {
    const si = to_i - from_i;
    const sj = to_j - from_j;
    const res = ndarray(new Array(si * sj), [si, sj]);
    for (let i = 0; i < si; i++)
        for (let j = 0; j < sj; j++)
            res.set(i, j, image.get(from_i + i, from_j + j));
    return res;
}

function crop(image: NdArray): NdArray {
    const ccs = get_ccs(image)
        //.filter(x => image.get(...x[0]) < 128)
        .sort((a, b) => b.length - a.length);
    const cc = ccs[0];
    const mini = Math.min(...cc.map(x => x[0]));
    const minj = Math.min(...cc.map(x => x[1]));
    const maxi = Math.max(...cc.map(x => x[0]));
    const maxj = Math.max(...cc.map(x => x[1]));
    return slice(image, mini, maxi + 1, minj, maxj + 1);
```

```
}

function get_verts(image: NdArray): [pos, pos, pos, pos] {
    function get_one_vert(i: (p: number, d: number) => number, j: (p: number, d:
    ↪  number) => number): pos {
        for (let d = 0; d < Math.max(...image.shape); d++) {
            for (let p = 0; p <= d; p++) {
                const i1 = i(p, d);
                const j1 = j(p, d);
                if (i1 < image.shape[0] && j1 < image.shape[1] && image.get(i1, j1) <
                ↪  128)
                    return [i1, j1];
            }
        }
        throw new InvalidArtagImageError("Oh shit!");
    }
    return [
        get_one_vert((p, d) => p, (p, d) => d - p),
        get_one_vert((p, d) => p, (p, d) => image.shape[1] - (d - p)),
        get_one_vert((p, d) => image.shape[0] - p, (p, d) => d - p),
        get_one_vert((p, d) => image.shape[0] - p, (p, d) => image.shape[1] - (d -
        ↪  p)),
    ];
}

function pos_sub(a: pos, b: pos): pos {
    return [a[0] - b[0], a[1] - b[1]];
}

function pos_add_one(a: pos, b: pos): pos {
    return [a[0] + b[0], a[1] + b[1]];
}

function pos_add(...args: pos[]): pos {
    return args.reduce((x, a) => pos_add_one(x, a), [0,0]);
}

function pos_mul(a: number, b: pos): pos {
    return [a * b[0], a * b[1]];
}

function get_points(verts: [pos, pos, pos, pos]): pos[] {
    const [p0, p1, p2, p3] = verts;
    const v1 = pos_sub(p1, p0);
    const v2 = pos_sub(p3, p1);
    const v3 = pos_sub(p2, p0);
    const v4 = pos_sub(p3, p2);
    const ib = pos_mul(0.5, pos_add(v1, v4));
    const jb = pos_mul(0.5, pos_add(v3, v2));
    function get_point(i: number, j: number): pos {
        i = (i + 0.5) / ARTAG_SIZE;
        j = (j + 0.5) / ARTAG_SIZE;
        return pos_add(p0, pos_mul(i, ib), pos_mul(j, jb));
    }
    const res: pos[] = [];
    for (let j = 0; j < ARTAG_SIZE; j++) {
        for (let i = 0; i < ARTAG_SIZE; i++)
            res.push(get_point(i, j));
    }
    return res;
```

```
}

function inplace_rotate(pixels: NdArray) {
    /* https://www.geeksforgeeks.org/inplace-rotate-square-matrix-by-90-degrees/ */
    const size = pixels.shape[0];
    const hs = (size / 2) | 0;
    for (let p = 0; p < hs; p++) {
        for (let k = 0; k < size - p * 2 - 1; k++) {
            const p0: pos = [p + k, p];
            const p1: pos = [size - p - 1, p + k];
            const p2: pos = [size - p - 1 - k, size - p - 1];
            const p3: pos = [p, size - p - 1 - k];
            const t = pixels.get(...p0);
            pixels.set(...p0, pixels.get(...p3));
            pixels.set(...p3, pixels.get(...p2));
            pixels.set(...p2, pixels.get(...p1));
            pixels.set(...p1, t);
        }
    }
}

function rotate_artag(pixels: NdArray) {
    let lc_marker_count = 0;
    for (let i = 0; i < 4; i++) {
        if (pixels.get(ARTAG_SIZE - 2, ARTAG_SIZE - 2))
            lc_marker_count++;
        inplace_rotate(pixels);
    }
    if (lc_marker_count != 1)
        throw new InvalidArtagImageError("expected to have 1 left-bottom-corner
        ↪  marker, but found " + lc_marker_count.toString());
    while (!pixels.get(ARTAG_SIZE - 2, ARTAG_SIZE - 2))
        inplace_rotate(pixels);
}

function sanitize_artag(pixels: NdArray) {
    for (let i = 0; i < 4; i++) {
        for (let j = 0; j < pixels.shape[0] - 1; j++) {
            if (pixels.get(j, 0) != 0)
                throw new InvalidArtagImageError("No border around artag");
        }
        inplace_rotate(pixels);
    }
}

/**
 * converts camera output to NdArray
 */
export function numbers_to_image(image: number[]): NdArray {
    const SIZE = [120, 160];
    const result = ndarray(new Array(SIZE[0] * SIZE[1]), SIZE);
    for (let i = 0; i < SIZE[0]; i++) {
        for (let j = 0; j < SIZE[1]; j++) {
            /* (historically) use only the green channel is used */
            const val = image[j + i * SIZE[1]]
            const green = (val >> 8) & 0xff;
            result.set(i, j, green);
        }
    }
    return result;
```

```
}
/**
 * converts input from tests to NdArray
 **/
export function string_to_image(str: string): NdArray {
    const SIZE = [120, 160];
    const splitted = str.trim().split(' ');
    const result = ndarray(new Array(SIZE[0] * SIZE[1]), SIZE);
    for (let i = 0; i < SIZE[0]; i++) {
        for (let j = 0; j < SIZE[1]; j++) {
            /* we (historically) use only the green channel */
            const val = splitted[j + i * SIZE[1]].slice(2, 4);
            result.set(i, j, parseInt(val, 16));
        }
    }
    return result;
}

export function read_artag(image: NdArray): NdArray {
    filter(image);
    image = slice(image, undefined, undefined, 4, undefined);
    otsu(image);
    image = crop(image);
    const verts = get_verts(image);
    const points = get_points(verts);
    const pixels = ndarray(points.map(x => image.get(...x.map(p => p | 0)) > 128 ? 1 :
    ↪  0), [ARTAG_SIZE, ARTAG_SIZE]);
    sanitize_artag(pixels);
    rotate_artag(pixels);
    return pixels;
}
```

solution/low_level_control/util/sensor_wrap.ts

```
import trik_brick from "trik_brick";
import { Timer } from "./timer";
import { setup_gyro, packed_uint_to_sec } from "./common";

export abstract class SensorWrap {
    abstract update(total_elapsed_time: number, delta_time: number): void;
}

export abstract class ScalarSensor extends SensorWrap {
    abstract get_value(): number;
}

export abstract class VectorSensor extends SensorWrap {
    abstract get_value(): number[];
}

export class DumbScalarSensor extends ScalarSensor {
    sensor: trik_brick.Sensor;
    current_reading: number = 0;
    constructor(sensor: trik_brick.Sensor, count: number) {
        super();
        this.sensor = sensor;
    }

    get_value(): number {
```

```
        return this.current_reading;
    }

    update(total_elapsed_time: number, delta_time: number): void {
        this.current_reading = this.sensor.read();
    }

}

export class MedianScalarSensor extends SensorWrap {
    sensor: trik_brick.Sensor
    buffer: number[];
    current_reading: number = 0;

    constructor(sensor: trik_brick.Sensor, count: number) {
        super();
        this.sensor = sensor;
        this.buffer = [];
        for (let i = 0; i < count; i++)
            this.buffer.push(sensor.read());
    }

    update(total_elapsed_time: number, delta_time: number): void {
        this.buffer.push(this.sensor.read());
        this.buffer.shift();
        const sorted = this.buffer.slice();
        sorted.sort();
        this.current_reading = sorted[((sorted.length + 1) / 2) | 0];
    }


    get_value(): number {
        return this.current_reading;
    }
}

export class SmoothUltrasonicSensorWrap extends ScalarSensor {
    sensor: trik_brick.Sensor
    previous_stable_reading: number;
    previous_unstable_reading: number;
    current_reading: number;

    constructor(sensor: trik_brick.Sensor) {
        super();
        this.sensor = sensor;
        this.current_reading = sensor.read();
        this.previous_stable_reading = this.current_reading;
        this.previous_unstable_reading = this.current_reading;
    }

    update(total_elapsed_time: number, delta_time: number): void {
        const currentRead = this.sensor.read();
        if (Math.abs(currentRead - this.previous_unstable_reading) > 10)
        {
            console.log("unstable: " + this.previous_stable_reading + ", " +
            ↪  currentRead);
            this.previous_unstable_reading = currentRead;
            this.current_reading = this.previous_stable_reading;
        }
```

```
        this.previous_unstable_reading = currentRead;
        this.previous_stable_reading = this.previous_unstable_reading;

        this.current_reading = currentRead;
    }


    get_value(): number {
        return this.current_reading;
    }
}

export interface GyroSensor extends SensorWrap {
    get_value(): number[];
    get_time(): number;
    get_pitch(): number;
    get_roll(): number;
    get_yaw(): number;
    translate(new_zero: number): ContinuousYawGyroSensor;
    reset(): void;
}

export interface ContinuousYawGyroSensor extends GyroSensor {}

export class RawGyroSensorWrap extends VectorSensor {
    sensor: trik_brick.GyroSensor;
    value: number[];
    timer: Timer;

    constructor(sensor: trik_brick.GyroSensor) {
        super();
        this.sensor = sensor;
        this.value = [0,0,0,0];
        setup_gyro(sensor);
        //sensor.newData.connect((read, time) => this.sensor_callback(read, time));
        this.timer = new Timer(() => this.sensor_callback(), 0.050);
        //calibrate_gyro(sensor);
        //console.log(sensor.getCalibrationValues());
    }

    sensor_callback(/*values: number[], time: number*/) {
        const MDEG_TO_RAD = Math.PI / 1000 / 180;
        const raw_value = this.sensor.read();
        this.value[0] = packed_uint_to_sec(raw_value[3]);
        this.value[1] = raw_value[4] * MDEG_TO_RAD;
        this.value[2] = raw_value[5] * MDEG_TO_RAD;
        this.value[3] = -raw_value[6] * MDEG_TO_RAD;
    }

    get_value(): number[] {
        return this.value;
    }

    get_time() { return this.value[0]; }
    get_pitch() { return this.value[1]; }
    get_roll() { return this.value[2]; }
    get_yaw() { return this.value[3]; }

    /** It's updated by the timer, so there is no point in doing anything here
     */
```

```
    update(total_elapsed_time: number, delta_time: number): void {}
};

class TranslatedGyroSensorWrap implements ContinuousYawGyroSensor {
    gyro: ContinuousYawGyroSensorWrap;
    zero: number;
    constructor(gyro: ContinuousYawGyroSensorWrap, zero: number) {
        this.gyro = gyro;
        this.zero = zero;
    }

    get_value(): number[] {
        let val = this.gyro.get_value().slice();
        val[3] -= this.zero;
        return val;
    }
    get_time(): number { return this.gyro.get_time(); }
    get_pitch(): number { return this.gyro.get_pitch(); }
    get_roll(): number { return this.gyro.get_roll(); }
    get_yaw(): number { return this.gyro.get_yaw() - this.zero; }
    translate(zero: number): ContinuousYawGyroSensor { return new
    ↪  TranslatedGyroSensorWrap(this.gyro, this.zero + zero); }
    update(total_elapsed_time: number, delta_time: number): void {
        this.gyro.update(total_elapsed_time, delta_time);
    }

    reset() {
        this.gyro.wrapped_yaw = this.zero;
        //this.zero = this.gyro.get_value()[3];
    }
}

/**
 * trik's gyro gives angles in range [-pi;pi], but it is more convienment to have
 ↪  "continuous" yaw axis (other axis do not matter anyways :shrug:)
 */
export class ContinuousYawGyroSensorWrap extends RawGyroSensorWrap implements
↪  ContinuousYawGyroSensor {
    previous_yaw: number = 0;
    wrapped_yaw: number = 0;
    static circle_delta(from: number, to: number) {
        /* gives a shortest by absolute value delta on unit circle */
        if (from < 0) from += Math.PI;
        if (to < 0) to += Math.PI;
        const d1 = to - from;
        const d2 = d1 + Math.PI;
        const d3 = d1 - Math.PI;
        const a1 = Math.abs(d1), a2 = Math.abs(d2), a3 = Math.abs(d3);
        if (a1 < a2) {
            if (a1 < a3)
                return d1;
            else
                return d3;
        } else if (a2 < a3)
            return d2;
        else
            return d3;
    }

    sensor_callback(/*values: number[], time: number*/) {
```

```
        super.sensor_callback();
        let val = this.value[3];
        let pval = this.previous_yaw;
        const delta = ContinuousYawGyroSensorWrap.circle_delta(pval, val);
        this.previous_yaw = this.value[3];
        this.wrapped_yaw += delta;
        // console.log(this.wrapped_yaw);
        this.value[3] = this.wrapped_yaw;
    }

    reset() {
        this.wrapped_yaw = 0;
    }

    translate(zero: number): ContinuousYawGyroSensor { return new
    ↪  TranslatedGyroSensorWrap(this, zero); }
}
```

solution/low_level_control/util/trik_wrap.ts

```
import trik_brick from "trik_brick";
import trik_script from "trik_script";

export function take_raw_image(): number[] {
    let image = trik_brick.getStillImage();
    if (image.length == 0)
        throw new Error("Camera is not available");
    return image;
}

export function take_filtered_image(): number[] {
    let image = trik_script.getPhoto();
    if (image.length == 0)
        throw new Error("Camera is not available");
    return image;
}
```

solution/low_level_control/util/common.ts

```
import trik_script from "trik_script";
import trik_brick, { GyroSensor } from "trik_brick";

export function delay(seconds: number) {
    trik_script.wait(seconds * 1000);
}

export function randint(lo: number, hi: number): number {
    return (Math.random() * (hi - lo) ) << 0;
}

export function get_time(): number {
    return new Date().getTime() / 1000;
}

export function beep(hz: number, ms: number) {
    trik_brick.playTone(hz, ms);
}
```

```
export function calibrate_gyro(gyro: GyroSensor) {
    const TIME = 30;
    gyro.calibrate(TIME * 1000);
    delay(TIME);
}

export function setup_gyro(gyro: GyroSensor) {
    const GYRO_FILE = "gyroscope.txt";
    let cal_text = trik_script.readAll(GYRO_FILE);
    if (cal_text.length === 0) {
        console.log("recalibrating gyroscope...");
        calibrate_gyro(gyro);
        const cal_val = gyro.getCalibrationValues();
        console.log("calibrated to " + JSON.stringify(cal_val));
        cal_text = [JSON.stringify(cal_val)];
        trik_script.removeFile(GYRO_FILE);
        trik_script.writeToFile(GYRO_FILE, cal_text[0]);
    }
    const cal_val = JSON.parse(cal_text[0]);

    gyro.setCalibrationValues(cal_val);
}

export function packed_uint_to_sec(val: number) {
    return val * Math.pow(2, 8) / 1000000;
}
```

solution/low_level_control/util/timer.ts

```
export class Timer {
    private id: number;

    constructor(callback: () => void, interval: number) {
        this.id = <any>setInterval(callback, interval * 1000);
    }

    clear() {
        clearInterval(<any>this.id);
    }
}
```

solution/low_level_control/util/pid.ts

```
export class PID {
    readonly p: number;
    readonly i: number;
    readonly d: number;

    integral: number;
    previous_error: number;

    constructor(p: number, i: number, d: number) {
        this.p = p;
        this.i = i;
        this.d = d;
        this.integral = 0;
```

```
        this.previous_error = 0;
    }

    get_next(setpoint:number, value: number, delta_time: number): number {
        const error = setpoint - value;
        this.integral += error * delta_time;
        const derivative = (error - this.previous_error) / delta_time;
        this.previous_error = error;
        const output = this.p * error + this.i * this.integral + this.d * derivative;
        return output;
    }

    reset() {
        this.previous_error = 0;
        this.integral = 0;
    }
}

export class LimitedPID extends PID {
    min: number;
    max: number;

    constructor(k_p: number, t_i: number, t_d: number, min: number, max: number) {
        super(k_p, t_i, t_d);
        this.min = min;
        this.max = max;
    }

    get_next(setpoint:number, value: number, delta_time: number): number {
        let val = super.get_next(setpoint, value, delta_time);
        if (val > this.max) val = this.max;
        if (val < this.min) val = this.min;
        return val;
    }
}
```

solution/low_level_control/low_level_interaction_proto.ts

```
import { WallsState } from "../FieldGraph";
import { Command } from "../Command";

export class LLCommand {
    public readonly xtype: string = 'Я - автор!';
    public result: LLResult | undefined;
}

export class ForwardCommand extends LLCommand {
    xtype = 'f';
    public amount: number = 0;
}
export class InspectWallsCommand extends LLCommand {
    xtype = 'w';
}
export class TurnCommand extends LLCommand {
    xtype = 't';
    public degrees: number = 0;
}
export class ScanArtagCommand extends LLCommand {
    xtype = 'a';
```

```
}
export class StopCommand extends LLCommand {
    xtype = 's';
}

export class LLResult {}

export class InspectWallsResult extends LLResult {
    public constructor(readonly walls: WallsState) { super(); }
}

export class ScanArtagResult extends LLResult {
    public constructor(readonly commands: Command[] | undefined) { super(); }
}
```

solution/low_level_control/low_level_interaction.ts

```
import { WallsState } from "../FieldGraph";
import ipc_entrypoint from "./movement";
import { LLCommand, LLResult, ForwardCommand, InspectWallsResult, InspectWallsCommand,
↪  TurnCommand, ScanArtagCommand, StopCommand, ScanArtagResult } from
↪  "./low_level_interaction_proto";

// Provides a bridge between low-level control and high-level controll

export class LLInteractionClient {
    constructor(private readonly low_level_loop_coro: Generator<void, void,
    ↪  LLCommand>) {
        console.log("LLInteractionClient constructor");
    }

    execute_command(command: LLCommand): LLResult | undefined {
        const res = this.low_level_loop_coro.next(command);
        if (res.done)
            throw new Error("low-level control stopped, but command was issued");
        //console.log("execute result is", command.result);
        return command.result;
    }

    execute_forward(amount: number) {
        const gg = new ForwardCommand();
        gg.amount = amount;
        this.execute_command(gg);
    }

    execute_inspect_walls(): WallsState {
        const result: InspectWallsResult =
        ↪  <InspectWallsResult>this.execute_command(new InspectWallsCommand());
        if (result === undefined) throw new Error("Result is undefined in
        ↪  execute_inspect_walls()")
        return result.walls;
    }

    execute_turn(degree: number) {
        const gg = new TurnCommand();
        gg.degrees = degree;
        this.execute_command(gg);
    }
```

```
    execute_scan_artag(): ScanArtagResult {
        return <ScanArtagResult>this.execute_command(new ScanArtagCommand());
    }

    execute_stop() {
        this.execute_command(new StopCommand());
    }
}

export function create_low_level_interation_client() {
    return new LLInteractionClient(ipc_entrypoint());
}
```

solution/low_level_control/robot_constants.ts

```
import trik_mailbox from 'trik_mailbox';

/* specific to construction */
export const WHEEL_DISTANCE = 0.175;
export const WHEEL_RADIUS = 0.03969885586091524; /* computed by moving trik and
↪  measuring encoders' values */ //0.0425
export const COUNTS_PER_REVOLUTION = 360;
export const COUNTS_PER_METER =
    COUNTS_PER_REVOLUTION / (2 * WHEEL_RADIUS * Math.PI);
export const METERS_PER_COUNT = 1 / COUNTS_PER_METER;
export const WHEEL_TO_CENTER_DISTANCE = 0.04;

export const CELL_FRONT_DISTANCE = 7.5;
export const MOVE_SPEED = 300;
export const ROTATE_SPEED = 200;
export const MOVE_SPEEDUP = 300;
export const ROTATE_SPEEDUP = 100;

/* specific to robot instance */
export const RIGHT_MOTOR_COEFFICIENT = 1;
export const LEFT_MOTOR_COEFFICIENT = 1;

/* some PID coefficients */
// 0.2   0.2 0     for non-accumulating speedy regulator
// 0.05  0   0.001 for accumulating speedy regulator
// 0.8   0.1 0     for positional regulator
// validated at 11.1824 V
export const MOTOR_PID_COEEFICIENTS = [0.8, 0.1, 0.005];

/* Reduces overshoot when rotating using gyro */
export let TRIK_90: number;
export let TRIK_180: number;

if (trik_mailbox.myHullNumber() == 0) {
    TRIK_90 =
        //0.475
        //0.5
        0.48 * Math.PI;
    TRIK_180 = 0.95 * Math.PI;
} else {
    TRIK_90 =
        //0.475
        0.485
        // 0.48
```

```
        * Math.PI;
    TRIK_180 =
        // 0.95
        1
        * Math.PI;
}
```

solution/low_level_control/movement/movement.ts

```
import { SmoothUltrasonicSensorWrap, ContinuousYawGyroSensorWrap, SensorWrap,
↪  ScalarSensor, ContinuousYawGyroSensor } from "../util/sensor_wrap";
import { make_motor, MotorController } from "./motors";
import { WHEEL_DISTANCE } from "../robot_constants";
import { PID } from "../util/pid";



// rotate k: 2.555

function cap_speed(speed: number) {
    return Math.sign(speed) * Math.min(500, Math.abs(speed));
}

export class Motors {
    readonly motor_left: MotorController;
    readonly motor_right: MotorController;
    constructor(motor_left: MotorController, motor_right: MotorController) {
        this.motor_left = motor_left;
        this.motor_right = motor_right;
    }
    update(left_speed: number, right_speed: number, delta_time: number) {
        this.motor_left.update(left_speed, delta_time);
        this.motor_right.update(right_speed, delta_time);
    }

    brake() {
        this.motor_left.brake();
        this.motor_right.brake();
    }
}

export class RulingMovementController  {
    readonly motors: Motors;
    static readonly D = WHEEL_DISTANCE;

    readonly speedup: number;
    constructor(motors: Motors, speedup: number) {
        this.motors = motors;
        this.speedup = speedup;
    }
    /**
     * Positive radius means turning right, negative - left, +-Infinity - forward
     */

    update(target_speed: number, rule_radius: number, total_elapsed_time: number,
    ↪  delta_time: number): void {
        target_speed = cap_speed(target_speed);
        let actual_target_speed = Math.sign(target_speed) *
        ↪  Math.min(Math.abs(target_speed), total_elapsed_time * this.speedup);
```

```
        if (rule_radius == Infinity || rule_radius == -Infinity)
        {
            this.motors.update(actual_target_speed, actual_target_speed, delta_time);
        }
        else
        {
            const R_m = Math.abs(rule_radius);
            const R_D = R_m + RulingMovementController.D / 2;
            const R = R_m - RulingMovementController.D / 2;
            const speed1 = cap_speed(actual_target_speed * R / R_m);
            const speed2 = cap_speed(actual_target_speed * R_D / R_m);

            //console.log(total_elapsed_time, actual_target_speed, speed1, speed2);

            // speed2 >= speed1

            if (rule_radius > 0)
                this.motors.update(speed2, speed1, delta_time);
            else
                this.motors.update(speed1, speed2, delta_time);
        }
    }

    brake() {
        this.motors.brake();
    }
}

export abstract class DirectionController {
    /** Returns desired curvature or null if don't have information
     */
    abstract update(target_speed: number, total_elapsed_time: number, delta_time:
    ↪  number): number | null;
    reset() {}
}

export class GyroDirectionController extends DirectionController {
    readonly gyro: ContinuousYawGyroSensor;
    static readonly coefficient: number = 20;

    constructor(gyro: ContinuousYawGyroSensor) {
        super();
        this.gyro = gyro;
    }
    update(target_speed: number, total_elapsed_time: number, delta_time: number):
    ↪  number | null {
        const error = this.gyro.get_yaw();
        const curvature = Math.sign(target_speed) * error *
        ↪  GyroDirectionController.coefficient;
        return curvature;
    }

    reset_gyro() {
        this.gyro.reset();
    }
}

export class WallProximityDirectionController extends DirectionController {
    readonly proximity: ScalarSensor;
    static readonly COEFFICIENT: number = 1;
```

```
    readonly coefficient: number;
    readonly target_proximity: number;
    readonly regulator: PID;
    error: number = 0;
    active: boolean = false;

    constructor(proximity: ScalarSensor, target_proximity: number, is_right: boolean)
    ↪  {
        super();
        this.proximity = proximity;
        this.regulator = new PID(0.4, 0, 0.1);
        if (!is_right)
            this.coefficient = -WallProximityDirectionController.COEFFICIENT;
        else
            this.coefficient = WallProximityDirectionController.COEFFICIENT;
        this.target_proximity = target_proximity;
    }

    update(target_speed: number, total_elapsed_time: number, delta_time: number):
    ↪  number | null {
        const prox_value = this.proximity.get_value();
        this.error = this.target_proximity - prox_value;
        //console.log(prox_value, error);
        this.active = prox_value < 25;
        if (!this.active)
            return null;
        const regulated_value = this.regulator.get_next(0, this.error, delta_time);
        return this.coefficient * regulated_value;
    }

    reset() {
        this.regulator.reset();
    }
}

export class CompositeDirectionController extends DirectionController {
    elements: [DirectionController, number][];

    constructor(elements: [DirectionController, number][]) {
        super();
        this.elements = elements;
    }

    update(target_speed: number, total_elapsed_time: number, delta_time: number):
    ↪  number | null {
        const results = this.elements
            .map((v) => [v[0].update(target_speed, total_elapsed_time, delta_time),
            ↪  v[1]])
            .filter((p): p is [number, number] => p[0] !== null);
        if (results.length == 0)
            return null;
        return results.reduce((a, v) => a + v[0] * v[1], 0) / results.reduce((a, v) =>
        ↪  a + v[1], 0);
    }

}

export class ForwardMovementController {
    readonly direction: CompositeDirectionController;
    readonly ruler: RulingMovementController;
```

```
static readonly coefficient: number = 20;
private previous_errors: number[] = [];
private readonly wall_controllers: WallProximityDirectionController[];
private readonly gyro_controller: GyroDirectionController;

constructor(ruler: RulingMovementController, direction:
↪  CompositeDirectionController) {
    this.direction = direction;
    this.ruler = ruler;
    this.wall_controllers =
    ↪  <WallProximityDirectionController[]>direction.elements.map(x =>
    ↪  x[0]).filter(x => x instanceof WallProximityDirectionController);
    this.gyro_controller = <GyroDirectionController>direction.elements.map(x =>
    ↪  x[0]).filter(x => x instanceof GyroDirectionController)[0];
}

private reset_gyro() {
    this.gyro_controller.reset_gyro();
    console.log("gyro reset");
}

private update_gyro_reset(total_elapsed_time: number) {
    const current_errors = this.wall_controllers.filter(x => x.active).map(x =>
    ↪  x.error);
    const ok = Math.max(...this.previous_errors.map(x =>
    ↪  Math.abs(x)).slice(undefined, -1)) < 3 && this.previous_errors.length >
    ↪  20;
    if (this.previous_errors.length >= 60 && ok) {
        console.log("we are stable for a pretty long time. resetting gyro...;
        ↪  previous_errors =", JSON.stringify(this.previous_errors))
        this.previous_errors = [];
        this.reset_gyro();
        return;
    }
    if (current_errors.length == 0) {
        if (this.previous_errors.length != 0) {
            console.log("we have lost walls! Were we moving ok?", (ok ? "YES;" :
            ↪  "no...;"), "previous_errors =",
            ↪  JSON.stringify(this.previous_errors));
            this.previous_errors = [];
            if (ok)
                this.reset_gyro();
        }
        return;
    }
    this.previous_errors.push(Math.max(...current_errors));
    //return false;
}

update(target_speed: number, total_elapsed_time: number, delta_time: number): void
↪  {
    const curvature = this.direction.update(target_speed, total_elapsed_time,
    ↪  delta_time) ?? 0;

    this.update_gyro_reset(total_elapsed_time);
    //console.log(target_speed, curvature, 1 / curvature);

    this.ruler.update(target_speed, 1 / curvature, total_elapsed_time,
    ↪  delta_time);
}
```

```
    brake(): void {
        this.ruler.brake();
    }
}

export class RotationMovementController {
    constructor(readonly motors: Motors, readonly speedup: number) {
        this.motors = motors;
        this.speedup = speedup;
    }

    update(target_speed: number, total_elapsed_time: number, delta_time: number): void
    ↪  {
        target_speed = cap_speed(target_speed);
        let actual_target_speed = Math.sign(target_speed) *
        ↪  Math.min(Math.abs(target_speed), total_elapsed_time * this.speedup);
        this.motors.update(actual_target_speed, -actual_target_speed, delta_time);
    }
    brake() {
        this.motors.brake();
    }
}
```

solution/low_level_control/movement/motors.ts

```
import trik_brick, { Encoder, encoder } from "trik_brick";

import { PID } from "../util/pid";
import { METERS_PER_COUNT } from "../robot_constants";

export abstract class MotorController {
    motor: trik_brick.Motor;
    encoder: trik_brick.Encoder;
    pid: PID;
    name: string;
    power_coefficient: number;
    constructor(name: string, power_coefficient: number, motor: trik_brick.Motor,
    ↪  encoder: trik_brick.Encoder, pid: PID) {
        this.name = name;
        this.motor = motor;
        this.encoder = encoder;
        this.pid = pid;
        this.power_coefficient = power_coefficient;

        encoder.reset();
    }

    protected validate(power: number): number {
        if (power > 100)
            power = 100;
        if (power < -100)
            power = -100;
        return power * this.power_coefficient;
    }

    protected set_motor_power(power: number) {
        this.motor.setPower(power * this.power_coefficient);
    }
```

```
    abstract update(target_speed: number, delta_time: number): void;

    brake() {
        this.motor.brake();
        this.pid.reset();
    }
}

export class SpeedyMotorController extends MotorController {
    last_encoder_position: number = 0;
    last_motor_output: number = 0;

    update(target_speed: number, delta_time: number) {
        const encoder_position = this.encoder.read();
        const encoder_velocity = (encoder_position - this.last_encoder_position) /
        ↪  delta_time;
        this.last_encoder_position = encoder_position;

        this.last_motor_output = this.last_motor_output +
        ↪  this.pid.get_next(target_speed, encoder_velocity, delta_time);
        this.last_motor_output = this.validate(this.last_motor_output);

        //console.log(0, this.name, target_speed, encoder_velocity,
        ↪  this.last_motor_output);

        this.set_motor_power(this.last_motor_output);
    }
}

export class PositionalMotorController extends MotorController {
    target_position: number = 0;

    update(target_speed: number, delta_time: number): void {
        const encoder_position = this.encoder.read();

        this.target_position = this.target_position + target_speed * delta_time;
        const new_power = this.pid.get_next(this.target_position, encoder_position,
        ↪  delta_time);

        //console.log(0, this.name, this.target_position, encoder_position,
        ↪  new_power);

        this.set_motor_power(this.validate(new_power));
    }

    brake() {
        super.brake();
        this.target_position = 0;
        this.encoder.reset();
    }
}

export class Encoders {
    left: Encoder;
    right: Encoder;
    constructor(left: Encoder, right: Encoder) {
        this.left = left;
        this.right = right;
    }
    get_position(): [number, number] {
```

```
        return [this.left.read(), this.right.read() ];
    }
    get_distance_from(position: [number, number]): number {
        const [pl, pr] = position;
        const [l, r] = this.get_position();
        const [dl, dr] = [METERS_PER_COUNT * (l - pl), METERS_PER_COUNT * (r - pr)];
        //console.log(pl, pr, l, r, dl, dr);
        return Math.abs(dl + dr) / 2;
    }
}

export function make_motor(name: string, port_number: string, power_coefficient:
↪  number, p: number, i: number, d: number): MotorController {
    const pid = new PID(p, i, d);
    const encoder = trik_brick.encoder("E" + port_number);
    const motor = trik_brick.motor("M" + port_number);
    return new PositionalMotorController(name, power_coefficient, motor, encoder,
    ↪  pid);
}

export function make_encoders(left_port: string, right_port: string) {
    const left = trik_brick.encoder("E" + left_port);
    const right = trik_brick.encoder("E" + right_port);
    return new Encoders(left, right);
}
```

solution/low_level_control/movement/index.ts

```
import trik_brick, { display } from "trik_brick";

import { get_time, delay } from "../util/common";
import { ScalarSensor, ContinuousYawGyroSensorWrap, MedianScalarSensor,
↪  ContinuousYawGyroSensor } from "../util/sensor_wrap";
import { make_motor, Encoders, make_encoders } from "./motors";
import { ForwardMovementController, GyroDirectionController, RulingMovementController,
↪  WallProximityDirectionController, CompositeDirectionController, Motors,
↪  RotationMovementController } from "./movement";
import { LEFT_MOTOR_COEFFICIENT, RIGHT_MOTOR_COEFFICIENT, TRIK_90, MOVE_SPEED,
↪  WHEEL_TO_CENTER_DISTANCE, ROTATE_SPEED, TRIK_180, CELL_FRONT_DISTANCE,
↪  MOTOR_PID_COEEFICIENTS, MOVE_SPEEDUP as MOVE_SPEEDUP, ROTATE_SPEEDUP } from
↪  "../robot_constants";
import { InspectWallsResult, ScanArtagCommand, LLCommand, ForwardCommand, LLResult,
↪  InspectWallsCommand, TurnCommand, StopCommand, ScanArtagResult } from
↪  "../low_level_interaction_proto";
import { take_filtered_image } from "../util/trik_wrap";
import { numbers_to_image, read_artag, InvalidArtagImageError } from
↪  "../../artag/libmorat-ng";
import { artag_decode } from "../../artag/artag_decode";

const MOTOR_LEFT_PORT = "2"
const MOTOR_RIGHT_PORT = "1"

const FRONT_SENSOR_PORT = "D1"
const BACK_SENSOR_PORT = "D2"
const LEFT_SENSOR_PORT = "A3"
const RIGHT_SENSOR_PORT = "A2"

abstract class MovementOperation {
    constructor(name: string) {
```

```
        this.name = name;
    }

    abstract update(total_elapsed_time: number, delta_time: number): boolean;
    readonly name: string;
}

class MoveToProximityOperation extends MovementOperation {
    readonly proximity_sensor: ScalarSensor;
    readonly target_distance: number;
    readonly movement_controller: ForwardMovementController;
    readonly target_speed: number;
    stop_counter = 0;


    constructor(target_speed: number, proximity_sensor: ScalarSensor,
    ↪  movement_controller: ForwardMovementController, target_distance: number, name:
    ↪  string) {
        super(name);

        this.proximity_sensor = proximity_sensor;
        this.movement_controller = movement_controller;
        this.target_distance = target_distance;
        this.target_speed = target_speed;
    }

    update(total_elapsed_time: number, delta_time: number): boolean {
        const current_distance = this.proximity_sensor.get_value()
        if (current_distance == -1)
            throw new Error("current_distance is -1");

        const error = Math.max(current_distance - this.target_distance, 0);

        //console.log(current_distance);

        //, error, regulated_target_speed, this.stop_counter);

        if (error < 2 || this.stop_counter > 0)
        {
            this.stop_counter += 1;
            this.movement_controller.update(0, total_elapsed_time, delta_time);
            if (this.stop_counter > 0)
            {
                this.movement_controller.brake();
                return false
            };
        }
        else
        {
            const regulated_target_speed = Math.sign(this.target_speed) *
            ↪  Math.min(Math.abs(this.target_speed), error * 10);
            this.movement_controller.update(regulated_target_speed,
            ↪  total_elapsed_time, delta_time);
        }
        return true;
    }
}

class RotateOperation extends MovementOperation {
```

```
    constructor(readonly target_speed: number, readonly controller:
    ↪  RotationMovementController, readonly gyro: ContinuousYawGyroSensor, name:
    ↪  string) {
        super(name);
    }

    update(total_elapsed_time: number, delta_time: number): boolean {
        const SLOWDOWN_BEGIN = Math.PI / 4;
        const error = this.gyro.get_yaw();
        const capped_error = Math.sign(error) * Math.min(Math.abs(error),
        ↪  SLOWDOWN_BEGIN);
        //console.log(error, capped_error);
        if (Math.abs(error) < Math.PI / 180) {
            this.controller.brake();
            return false;
        }
        this.controller.update(capped_error * this.target_speed / SLOWDOWN_BEGIN,
        ↪  total_elapsed_time, delta_time);
        return true;
    }
}


class MoveMetersOperation extends MovementOperation {
    readonly movement_controller: ForwardMovementController;
    readonly target_speed: number;
    readonly initial_position: [number, number];
    readonly target_distance: number;
    readonly encoders: Encoders;

    constructor(target_speed: number, movement_controller: ForwardMovementController,
    ↪  encoders: Encoders, distance: number, name: string) {
        super(name);
        this.movement_controller = movement_controller;
        this.target_speed = target_speed;
        this.initial_position = encoders.get_position();
        this.target_distance = distance;
        this.encoders = encoders;
    }

    update(total_elapsed_time: number, delta_time: number): boolean {
        const position = this.encoders.get_distance_from(this.initial_position);
        const error = this.target_distance - position;
        if (error < 0.005) {
            this.movement_controller.update(0, total_elapsed_time, delta_time);
            this.movement_controller.brake();
            return false;
        }

        const regulated_target_speed = Math.sign(this.target_speed) *
        ↪  Math.min(Math.abs(this.target_speed), error * 10000);

        //console.log(error, regulated_target_speed);
        this.movement_controller.update(regulated_target_speed, total_elapsed_time,
        ↪  delta_time);

        return true;
    }
}
```

```
class DelayOperation extends MovementOperation {
    constructor(readonly delay: number, name: string) {
        super(name);
    }

    update(total_elapsed_time: number, delta_time: number): boolean {
        return total_elapsed_time < this.delay;
    }
}


let [p, i, d] = MOTOR_PID_COEEFICIENTS;
const motor_left = make_motor("L", MOTOR_LEFT_PORT, LEFT_MOTOR_COEFFICIENT, p, i, d);
↪  // new SpeedyMotorWrapper("L", motor1, encoder1, pid1);
const motor_right = make_motor("R", MOTOR_RIGHT_PORT, RIGHT_MOTOR_COEFFICIENT, p, i,
↪  d); // new SpeedyMotorWrapper("R", motor2, encoder2, pid2);
const motors = new Motors(motor_left, motor_right);

const encoders = make_encoders(MOTOR_LEFT_PORT, MOTOR_RIGHT_PORT);

const front_sensor = new MedianScalarSensor(trik_brick.sensor(FRONT_SENSOR_PORT), 3);
const back_sensor = new MedianScalarSensor(trik_brick.sensor(BACK_SENSOR_PORT), 3);
const left_sensor = new MedianScalarSensor(trik_brick.sensor(LEFT_SENSOR_PORT), 3);
const right_sensor = new MedianScalarSensor(trik_brick.sensor(RIGHT_SENSOR_PORT), 3);


let gyro: ContinuousYawGyroSensor = new
↪  ContinuousYawGyroSensorWrap(trik_brick.gyroscope());
function make_forward_movement_controller() {
    const ruler = new RulingMovementController(motors, MOVE_SPEEDUP);
    const direction_rwall = new WallProximityDirectionController(right_sensor, 13.5,
    ↪  true);
    const direction_lwall = new WallProximityDirectionController(left_sensor, 13.5,
    ↪  false);
    const direction_gyro = new GyroDirectionController(gyro);
    const composite_ruler = new CompositeDirectionController([[direction_gyro, 0.2],
    ↪  [direction_rwall, 0.4], [direction_lwall, 0.4]]);
    const current_controller = new ForwardMovementController(ruler, composite_ruler);
    return current_controller;
}

function* make_move_until(speed: number, proximity: number, name: string) {
    yield new MoveToProximityOperation(speed, speed < 0 ? back_sensor : front_sensor,
    ↪  make_forward_movement_controller(), proximity, name);
}

function* make_move_to(speed: number, distance: number, name: string) {
    yield new MoveMetersOperation(speed, make_forward_movement_controller(), encoders,
    ↪  distance, name);
}

function* make_raw_rotation(speed: number, rotation: number, name: string) {
    yield new RotateOperation(speed, new RotationMovementController(motors,
    ↪  ROTATE_SPEEDUP), gyro.translate(-rotation), name);
}

function* make_noop(name: string) {
    yield new DelayOperation(1 / 60 + 0.01, name);
}
```

```
function* definite_proximity(sensor: ScalarSensor[]): Generator<MovementOperation,
↪  [number, number][]> {
    /* TODO: maybe add some filtering */
    const vv = [];
    for (let i = 0; i < 5; i++) {
        vv.push(sensor.map(x => x.get_value()));
        yield* make_noop("definite_proximity_yield");
    }

    console.log("vv is", JSON.stringify(vv));
    let res: [number, number][] = new Array(vv[0].length);
    for (let i = 0; i < res.length; i++)
        res[i] = [0, 0];
    for (let i = 0; i < vv.length; i++)
        for (let j = 0; j < vv[i].length; j++)
            res[j][0] += vv[i][j];

    res = res.map(x => [x[0] / vv.length, x[1]]);

    for (let i = 0; i < vv.length; i++)
        for (let j = 0; j < vv[i].length; j++)
            res[j][1] += Math.pow(res[j][0] - vv[i][j], 2);
    res = res.map(x => [x[0], x[1] / vv.length]);

    console.log("res is", JSON.stringify(res));
    return res;
}

function* make_cell_rotation(rotation: number, name: string) {
    rotation = rotation | 0;
    let degree = 0;
    switch (rotation) {
        case 0:
            degree = 0;
            break;
        case 1:
            degree = TRIK_90;
            break;
        case 2:
            degree = TRIK_180;
            break;
        case 3:
            degree = -TRIK_90;
            break;
    }
    yield* make_move_to(ROTATE_SPEED, WHEEL_TO_CENTER_DISTANCE,
    ↪  "move_forward_before_rotation");
    yield* make_raw_rotation(ROTATE_SPEED, degree, "rotate");
    gyro = gyro.translate(-degree);
    yield* make_move_to(-ROTATE_SPEED, WHEEL_TO_CENTER_DISTANCE,
    ↪  "move_backwards_after_rotation");
}

function* make_onwards_one(): Iterable<MovementOperation> {
    const f = (yield* definite_proximity([front_sensor]))[0][0];
    console.log("Forward distance:", f);
    if (f < 20 && f != -1) throw Error("Won't move into wall");
    else if (f - 40 < 15) yield* make_move_until(MOVE_SPEED, CELL_FRONT_DISTANCE,
    ↪  "move_forward_until_wall");
    else yield* make_move_to(MOVE_SPEED, 0.4, "move_forward_tacho");
```

```
}

function inspect_wall(distance: number, error: number, is_infrared: boolean): number |
↪  undefined {
    if (is_infrared) {
        if (distance < 35)
            return 0;
        else
            return 1;
    } else {
        if (error > 5)
            return undefined;
        if (distance == -1)
            throw new Error("distance is -1");
        if (distance < 30)
            return 0;
        //if (distance < 70)
            return 1;
        //if (distance < 100)
        //    return 2;
        //return 3;
    }
}

function* make_onwards_many(count: number): Iterable<MovementOperation> {
    if (count == 1)
        yield* make_onwards_one();
    else {
        const f = (yield* definite_proximity([front_sensor]))[0];
        console.log("Forward distance:", f);
        const free = inspect_wall(f[0], f[1], false) ?? 0;
        if (f[0] < 20 && f[0] != -1) throw Error("Won't move into wall");
        if (free == count && free < 2)
            yield* make_move_until(MOVE_SPEED, CELL_FRONT_DISTANCE,
            ↪  "move_forward_until_wall");
        else
            yield* make_move_to(MOVE_SPEED, 0.4 * count, "move_forward_tacho");
    }
}

function* inspect_walls(): Generator<MovementOperation, InspectWallsResult> {
    //const aaa: [number, boolean][] = [];
    const sens = [front_sensor, back_sensor, left_sensor, right_sensor];
    const bbb = yield* definite_proximity(sens);
    const ccc: [number, number, boolean][] = new Array(bbb.length);
    for (let i = 0; i < sens.length; i++) {
        ccc[i] = [bbb[i][0], bbb[i][1], sens[i] == left_sensor || sens[i] ==
        ↪  right_sensor];
    }

    console.log("inspect_walls:", bbb);
    let [f, b, l, r] = ccc.map(x => inspect_wall(x[0], x[1], x[2]));
    if (f == undefined || b == undefined)
        return yield* inspect_walls();
    return new InspectWallsResult({
        f_free: f,
        b_free: b,
        l_free: l ?? 1,
        r_free: r ?? 1,
        // f_wall: f < 2 ? f : undefined,
```

```
        // b_wall: b < 2 ? b : undefined,
        // l_wall: (l ?? 1) < 1 ? l : undefined,
        // r_wall: (r ?? 1) < 1 ? r : undefined,
    });
}

function scan_artag(): ScanArtagResult {
    console.log("scanning artag...");
    const raw_image = take_filtered_image();
    display().show(raw_image, 160, 120, "rgb32");
    const image = numbers_to_image(raw_image);
    try {
        const data = read_artag(image);
        const commands = artag_decode(data);
        return new ScanArtagResult(commands);
    } catch (e) {
        if (e instanceof InvalidArtagImageError)
            return new ScanArtagResult(undefined);
        throw e;
    }
}



function* ipc_logic(): Generator<MovementOperation | undefined, void, LLCommand |
↪  undefined> {
    console.log("ipc_logic");
    while (true) {
        const command = yield;

        console.log("got command (2):", command);

        if (command instanceof ForwardCommand) {
            for (const p of make_onwards_many(command.amount))
                yield p;
            command.result = new LLResult();
        } else if (command instanceof InspectWallsCommand) {
            yield* make_noop("yield_for_sensors");
            command.result = yield* inspect_walls();
        } else if (command instanceof TurnCommand) {
            const magic = (((-command.degrees / 90) | 0) + 4) % 4;
            yield* make_cell_rotation(magic, "rotate_" + magic.toString());
            command.result = new LLResult();
        } else if (command instanceof ScanArtagCommand) {
            yield* make_noop("yield_for_sensors");
            command.result = scan_artag();
        } else if (command instanceof StopCommand) {
            command.result = new StopCommand();
            break;
        } else {
            throw new Error("Got unknown command");
        }
    }
}

/* some coroutine spaghetti is going on. But it is better than having two threads */
function* execute_logic(ipc_logic_generator: Generator<MovementOperation | undefined,
↪  void, LLCommand | undefined>): Generator<void, void, LLCommand>
{
```

```
    const logic = ipc_logic_generator[Symbol.iterator]();
    let current_operation: MovementOperation | null = null;
    let current_command: LLCommand = yield;
    logic.next(current_command);
    //console.log("got initial command", current_command)

    let prev_time = get_time();
    let start_time = prev_time;
    delay(0.01);

    while (true) {
        const time = get_time();
        const delta_time = time - prev_time;
        const total_elapsed_time = time - start_time;

        prev_time = time;

        front_sensor.update(total_elapsed_time, delta_time);
        back_sensor.update(total_elapsed_time, delta_time);
        left_sensor.update(total_elapsed_time, delta_time);
        right_sensor.update(total_elapsed_time, delta_time);
        gyro.update(total_elapsed_time, delta_time);

        if (current_operation == null) {
            const value = logic.next(current_command);
            //console.log("got operation", value.value)
            if (value.done)
                break;
            if (value.value == undefined) {
                //console.log("yielding to high level...")
                current_command = yield;
                //console.log("got:", current_command)
                continue;
            }
            start_time = prev_time = get_time();
            current_operation = value.value!;
            delay(0.01);
            console.log("new operation started:", start_time, current_operation.name);
            continue;
        }
        if (!current_operation.update(total_elapsed_time, delta_time)) {
            console.log("Operation '" + current_operation.name + "' has Completed");
            current_operation = null;
        }

        delay(1 / 60);
    }
    console.log("Control loop exited");
}

/* Just in case */
delay(1);

//execute_logic(test_logic());
//execute_logic(left_handle_rule_logic());

function ipc_entrypoint(): Generator<void, void, LLCommand> {
    console.log("worker entry");
    const co = execute_logic(ipc_logic());
    co.next();
```

```
    return co;
}

export default ipc_entrypoint;
```

solution/entry_points/day4/real_1.ts

```
import '../../../shim-global';

import brick from 'trik_brick';
import script from 'trik_script';
import mailbox from 'trik_mailbox';
import { Direction, negateDirection, Node } from '../../FieldGraph';
import { RobotReal } from '../../robots/RobotReal';
import { delay } from '../../low_level_control/util/common';
import {
    run_generator,
    normalizeAngle,
    normalizeDirection,
} from '../../helpers';
import { Command } from '../../Command';

const FIRST_ROBOT_HULL_NUM = 0;
const FIRST_ROBOT_DIR = Direction.Up;

const SECOND_ROBOT_HULL_NUM = 1;
const SECOND_ROBOT_DIR = Direction.Left;

switch (mailbox.myHullNumber()) {
    case FIRST_ROBOT_HULL_NUM: {
        const robot = new RobotReal(FIRST_ROBOT_DIR);
        const localMapStart = robot.localNode;

        mailbox.receive(true); // wait for the second robot finish scanning artag &
        ↪ moving

        robot.scan();
        let relative_dir: Direction;
        let scanned_node: Node;

        for (const cell of robot.walk()) {
            run_generator(robot.moveToLocalMapNode(cell));

            console.log('sending stuff to', SECOND_ROBOT_HULL_NUM);
            mailbox.send(
                SECOND_ROBOT_HULL_NUM,
                JSON.stringify({ type: 'rescan' })
            );
            const msg = JSON.parse(mailbox.receive(true));
            if (msg !== null) {
                console.log('WHOOOOP!', JSON.stringify(msg));
                const { dwall }: { dwall: Direction } = msg;
                relative_dir = negateDirection(dwall);
                scanned_node = robot.localNode;

                break; // from now on we should use optimal localization algorithm
            }
        }
```

```
robot.localize();

console.log('localized');

let localCell: Node; // second robot coords in the first robot local
↪  coordinate system
switch (relative_dir!) {
    case Direction.Right:
        localCell = [scanned_node![0], scanned_node![1] + 1];
        break;
    case Direction.Up:
        localCell = [scanned_node![0] - 1, scanned_node![1]];
        break;
    case Direction.Left:
        localCell = [scanned_node![0], scanned_node![1] - 1];
        break;
    case Direction.Down:
        localCell = [scanned_node![0] + 1, scanned_node![1]];
        break;
}

run_generator(robot.moveNearOccupiedCell(localCell!));

brick.display().clear();
brick.display().addLabel(`(${robot.xReal},${robot.yReal})`, 1, 1);
brick.display().redraw();
brick.playTone(1000, 100);

delay(10);

mailbox.send(
    SECOND_ROBOT_HULL_NUM,
    JSON.stringify({
        type: 'coords',
        args: { node: localCell!, dx: robot.dx, dy: robot.dy },
    })
);

const secondStart: Node = JSON.parse(mailbox.receive(true));
const secondStartLocal: Node = [
    secondStart[0] + robot.dy!,
    secondStart[1] + robot.dx!,
];

console.log(
    'Starts:',
    JSON.stringify([secondStartLocal, localMapStart])
);

const dfirst =
    Math.abs(robot.yLocal - localMapStart[0]) +
    Math.abs(robot.xLocal - localMapStart[1]);
const dsecond =
    Math.abs(robot.yLocal - secondStartLocal[0]) +
    Math.abs(robot.xLocal - secondStartLocal[1]);

if (dfirst <= dsecond) {
    try {
        run_generator(robot.moveToLocalMapNode(localMapStart));
    } catch (e) {
```

```
            if (e instanceof RangeError) {
                run_generator(robot.moveToLocalMapNode(secondStartLocal));
            } else throw e;
        }
    } else {
        try {
            run_generator(robot.moveToLocalMapNode(secondStart));
        } catch (e) {
            if (e instanceof RangeError) {
                run_generator(robot.moveToLocalMapNode(localMapStart));
            } else throw e;
        }
    }

    mailbox.send(
        SECOND_ROBOT_HULL_NUM,
        JSON.stringify({
            type: 'stop',
            args: null,
        })
    );

    brick.display().clear();
    brick.display().addLabel('finish', 1, 20);
    brick.display().redraw();
    delay(10);
    break;
}
case SECOND_ROBOT_HULL_NUM: {
    /* oh well */
    mailbox.connect('192.168.77.1');

    const robot = new RobotReal(SECOND_ROBOT_DIR, false);

    const secondStart = robot.localNode;

    const dir_initial = robot.direction;
    const commands = robot.findArtag();
    if (commands === undefined) throw new Error('Cannot read artag!!1');
    const dir_delta = normalizeDirection(dir_initial - robot.direction);
    const rot_cmds = Array(dir_delta).fill(Command.TurnLeft);
    run_generator(robot.runCommands(rot_cmds));

    // run_generator(robot.runCommandsRaw(commands));
    run_generator(robot.runCommands(commands));

    mailbox.send(FIRST_ROBOT_HULL_NUM, 'null');

    brick.playTone(1000, 100);

    brick.display().clear();
    brick.display().addLabel('finish', 1, 20);
    brick.display().redraw();

    const walls_initial = robot.get_walls_neighbors();
    let wait = true;

    while (wait) {
        const messageJson = mailbox.receive(true);
        const message: { type: string; args: any } = JSON.parse(
```

```
    messageJson
);
switch (message.type) {
    case 'rescan': {
        const walls = robot.get_walls_neighbors();
        // only one wall can change
        const newWalls = walls.filter(
            (x) => !walls_initial.includes(x)
        );
        const removedWalls = walls_initial.filter(
            (x) => !walls.includes(x)
        );
        let dwall: Direction | undefined = undefined;
        if (newWalls.length) {
            dwall = newWalls[0];
        } else if (removedWalls.length) {
            dwall = newWalls[0];
        }

        if (dwall !== undefined) {
            mailbox.send(
                FIRST_ROBOT_HULL_NUM,
                JSON.stringify({
                    dwall,
                })
            );
        } else mailbox.send(FIRST_ROBOT_HULL_NUM, 'null');
        break;
    }

    case 'coords': {
        const {
            args: { node, dx, dy },
        }: {
            args: { node: Node; dx: number; dy: number };
        } = message;

        // local_first = real + d
        // real = local_first - d
        // d_second = local_second - real

        const yReal = node[0] - dy;
        const xReal = node[1] - dx;

        robot.dx = robot.localNode[1] - yReal;
        robot.dy = robot.localNode[0] - xReal;

        console.log(
            'sending to first: ',
            JSON.stringify([
                secondStart[0] - robot.dy,
                secondStart[1] - robot.dx,
            ])
        );
        mailbox.send(
            FIRST_ROBOT_HULL_NUM,
            JSON.stringify([
                secondStart[0] - robot.dy,
                secondStart[1] - robot.dx,
            ])
```

```
                    );

                    break;
                }

                case 'stop': {
                    wait = false;
                    break;
                }
            }
        }
        break;
    }
    default: {
        throw new Error(`Unknown hull number: ${mailbox.myHullNumber()}`);
    }
}
```

solution/robots/Robot.ts

```
import { Command } from '../Command';
import { normalizeDirection, run_generator } from '../helpers';
import {
    FieldGraph,
    FIELD_SIZE,
    Direction,
    WallsState,
    Node,
    CellState,
} from '../FieldGraph';

export abstract class Robot {
    private readonly verbose = true;
    private localization_data = {
        xmax: 0,
        ymax: 0,
        xmin: FIELD_SIZE,
        ymin: FIELD_SIZE,
    };

    protected readonly direction_initial: Direction;

    constructor(
        public direction: Direction,
        public autoScanField: boolean = true
    ) {
        this.direction_initial = direction;
    }

    protected localMap = new FieldGraph();

    public yLocal = FIELD_SIZE;
    public xLocal = FIELD_SIZE;
    public get localNode(): Node {
        return [this.yLocal, this.xLocal];
    }

    public get realNode(): Node {
        return [this.yReal, this.xReal];
```

```
}

public dx?: number = undefined;
public dy?: number = undefined;

public get located() {
    return this.dx !== undefined && this.dy !== undefined;
}

get xReal() {
    if (!this.located) throw new Error('Not located yet');
    return this.xLocal - this.dx!;
}

get yReal() {
    if (!this.located) throw new Error('Not located yet');
    return this.yLocal - this.dy!;
}

/**
 * Move forward by `n` cells
 */
protected abstract _forward(n?: number): void;

/**
 * Move forward by n cells
 * Yields the position after each rescan
 */
public *forward(n: number = 1): Iterable<Node> {
    if (this.direction === Direction.Up) this.yLocal -= n;
    else if (this.direction === Direction.Down) this.yLocal += n;
    else if (this.direction === Direction.Left) this.xLocal -= n;
    else if (this.direction === Direction.Right) this.xLocal += n;
    this._forward(n);
    if (this.autoScanField) {
        this.scan();
        this.localize_iteration();
    }

    yield this.localNode;
}

/**
 * Turn left by `deg` degrees
 */
public abstract turn(deg: number): void;

/**
 * Get the number of free cells at each direction:
 *   - `r:` right
 *   - `l:` left
 *   - `f:` forward
 *
 * If counting of free cells is impossible, the method should output
 * whether the wall at the direction exist (0 if exist, 1 otherwise)
 *
 * If you can't say anything about the walls, set the corresponding property to
 ↪  `undefined`
 */
public abstract getWalls(): WallsState;
```

```
public get_walls_neighbors(): Direction[] {
    const ret: Direction[] = [];
    const w = this.getWalls();
    const ifWall = (x?: number) => x !== undefined && x === 0;
    switch (this.direction) {
        case Direction.Right:
            if (ifWall(w.f_free)) {
                ret.push(Direction.Right);
            }
            if (ifWall(w.l_free)) {
                ret.push(Direction.Up);
            }
            if (ifWall(w.b_free)) {
                ret.push(Direction.Left);
            }
            if (ifWall(w.r_free)) {
                ret.push(Direction.Down);
            }
            break;
        case Direction.Up:
            if (ifWall(w.r_free)) {
                ret.push(Direction.Right);
            }
            if (ifWall(w.f_free)) {
                ret.push(Direction.Up);
            }
            if (ifWall(w.l_free)) {
                ret.push(Direction.Left);
            }
            if (ifWall(w.b_free)) {
                ret.push(Direction.Down);
            }
            break;
        case Direction.Left:
            if (ifWall(w.b_free)) {
                ret.push(Direction.Right);
            }
            if (ifWall(w.r_free)) {
                ret.push(Direction.Up);
            }
            if (ifWall(w.f_free)) {
                ret.push(Direction.Left);
            }
            if (ifWall(w.l_free)) {
                ret.push(Direction.Down);
            }
            break;
        case Direction.Down:
            if (ifWall(w.l_free)) {
                ret.push(Direction.Right);
            }
            if (ifWall(w.b_free)) {
                ret.push(Direction.Up);
            }
            if (ifWall(w.r_free)) {
                ret.push(Direction.Left);
            }
            if (ifWall(w.f_free)) {
                ret.push(Direction.Down);
```

```
            }
            break;
    }

    return ret;
}

/**
 * Turns left
 * Yields the position after each rescan
 */
public *turnLeft(n: number = 1): Iterable<Node> {
    this.turn(90 * n);
    this.direction = normalizeDirection(this.direction + n);
    if (this.autoScanField) {
        this.scan();
        this.localize_iteration();
    }

    yield this.localNode;
}

/**
 * Turns right
 * Yields the position after each rescan
 */
public *turnRight(n: number = 1): Iterable<Node> {
    yield* this.turnLeft(-n);
}

public *runCommandsRaw(commands: Command[]): Iterable<Node> {
    for (const command of commands) {
        yield* this.runCommands([command]);
    }
}

public *runCommands(commands: Command[]): Iterable<Node> {
    let i = 0;
    while (i < commands.length) {
        switch (commands[i]) {
            case Command.MoveForward:
                let fwdCounter = 1;
                i += 1;
                while (
                    commands[i] == Command.None ||
                    commands[i] == Command.MoveForward
                ) {
                    if (commands[i] == Command.MoveForward) fwdCounter += 1;
                    i += 1;
                }

                yield* this.forward(fwdCounter);
                break;

            case Command.None:
                i += 1;
                break;

            case Command.TurnLeft:
            case Command.TurnRight:
```

```
                let rotCounter = 0;
                while (
                    commands[i] == Command.TurnLeft ||
                    commands[i] == Command.TurnRight ||
                    commands[i] == Command.None
                ) {
                    if (commands[i] == Command.TurnLeft) {
                        rotCounter += 1;
                    } else if (commands[i] == Command.TurnRight) {
                        rotCounter -= 1;
                    }
                    i += 1;
                }

                yield* this.turnLeft(rotCounter);
                break;
            }
        }
    }

    public scan(): void {
        for (const node of this.localMap.updateField(
            this.localNode,
            this.getWalls(),
            this.direction
        )) {
            this.localize_iteration(node);
        }

        if (this.verbose) {
            console.log('Map:');
            console.log(this.localMap);
        }
    }

    private localize_iteration([y, x]: Node = this.localNode) {
        let { xmin, xmax, ymin, ymax } = this.localization_data;

        ymin = Math.min(ymin, y);
        ymax = Math.max(ymax, y);
        // if (this.verbose) console.log('ymin, ymax', ymin, ymax);
        xmin = Math.min(xmin, x);
        xmax = Math.max(xmax, x);
        // if (this.verbose) console.log('xmin, xmax', xmin, xmax);

        if (ymax - ymin + 1 === FIELD_SIZE) {
            this.dy = ymin;

            // set the borders of the field when found Y coordinate
            for (let jj = 0; jj < this.localMap.length; ++jj) {
                this.localMap.field[ymin - 1][jj] = CellState.Wall;
                this.localMap.field[ymax + 1][jj] = CellState.Wall;
            }
        }

        if (xmax - xmin + 1 === FIELD_SIZE) {
            this.dx = xmin;

            // set the borders of the field when found X coordinate
            for (let ii = 0; ii < this.localMap.length; ++ii) {
```

```
            this.localMap.field[ii][xmin - 1] = CellState.Wall;
            this.localMap.field[ii][xmax + 1] = CellState.Wall;
        }
    }

    this.localization_data = { xmin, xmax, ymin, ymax };
}

/**
 * Walk all cells of the local map
 */
public *walk(): Iterable<Node> {
    yield* this.localMap.dfs(this.localNode);
}

public localize(fullMapScan = false): void {
    this.scan();
    for (const [i, j] of this.walk()) {
        const hasUnscannedNeighbor = () =>
            [...this.localMap.neighbors([i, j], true)].reduce(
                (has, [i, j]) =>
                    has || this.localMap.field[i][j] === CellState.Unknown,
                false
            );

        if (hasUnscannedNeighbor()) {
            for (const _ of this.moveToLocalMapNode([i, j])) {
                if (this.located && !fullMapScan) return;
                if (!hasUnscannedNeighbor()) break;
            }
        }

        if (this.located && !fullMapScan) return;
    }
}

private moveByAbsoluteDirectionCommands(
    to: Direction,
    currentDirection: Direction = this.direction
): Command[] {
    const diff = normalizeDirection(to - currentDirection);
    const commands: Command[] = [];
    // for (let i = 0; i < diff; ++i) commands.push(Command.TurnLeft);
    switch (diff) {
        case 0:
            break;
        case 1:
            commands.push(Command.TurnLeft);
            break;
        case 2: // Rotate by 180°
            commands.push(Command.TurnLeft, Command.TurnLeft);
            break;
        case 3:
            commands.push(Command.TurnRight);
            break;
    }
    commands.push(Command.MoveForward);

    return commands;
}
```

```
public *moveByAbsoluteDirection(to: Direction): Iterable<Node> {
    yield* this.runCommands(this.moveByAbsoluteDirectionCommands(to));
}

private moveToNeighborNodeCommands(
    to: Node,
    currentPosition: Node,
    currentDirection: Direction = this.direction
): Command[] {
    const rel = this.localMap.relativePosition(to, currentPosition);
    if (rel === Direction.Unknown)
        throw new Error("Can't move to node " + JSON.stringify(to));
    return this.moveByAbsoluteDirectionCommands(rel, currentDirection);
}

public *moveToNeighborNode(to: Node): Iterable<Node> {
    yield* this.runCommands(
        this.moveToNeighborNodeCommands(to, this.localNode)
    );
}

/**
 * Converts path to robot commands
 * @param path The required path. Should start from current node.
 * @param currentDirection
 */
private pathToCommands(
    path: Node[],
    currentDirection = this.direction
): Command[] {
    let commands: Command[] = [];
    let currentPosition = path[0];
    for (const nextNode of path.slice(1)) {
        commands = commands.concat(
            this.moveToNeighborNodeCommands(
                nextNode,
                currentPosition,
                currentDirection
            )
        );
        currentDirection = this.localMap.relativePosition(
            nextNode,
            currentPosition
        );
        currentPosition = nextNode;
    }
    return commands;
}

private shortestCommandSetMetric(path: Command[]): number {
    const turns = path.reduce(
        (acc, command) =>
            command === Command.TurnLeft || command === Command.TurnRight
                ? acc + 1
                : 0,
        0
    );
    return path.length + turns;
}
```

```
public getReachableCellNearOccupiedCell(cell: Node): Node | undefined {
    const neighbors = [...this.localMap.neighbors(cell, true)];

    const ret = neighbors.reduce<{ neighbor?: Node; plen: number }>(
        (acc, neighbor) => {
            const { known, path } = this.localMap.bfs_path(
                this.localNode,
                neighbor
            );
            if (!known || path === undefined) return acc;

            const plen = this.shortestCommandSetMetric(
                this.pathToCommands(path)
            );
            if (plen < acc.plen) {
                return { plen, neighbor };
            }

            return acc;
        },
        { neighbor: undefined, plen: Infinity }
    );

    return ret.neighbor;
}

public *moveNearOccupiedCell(cell: Node): Iterable<Node> {
    if (this.verbose) console.log('Moving near:', cell);
    const known_neighbor = this.getReachableCellNearOccupiedCell(cell);
    if (known_neighbor !== undefined) {
        yield* this.moveToLocalMapNode(known_neighbor);
        return;
    }

    const unknown_neighbors = [...this.localMap.neighbors(cell, true)];
    for (const node of unknown_neighbors) {
        console.log('unknown neighbor: ', node);

        try {
            yield* this.moveToLocalMapNode(node);
        } catch (e) {
            if (e instanceof RangeError) {
            } else throw e;
        }

        console.log('localNode: ', this.localNode);

        if (
            this.localMap.relativePosition(this.localNode, cell) !==
            Direction.Unknown
        )
            return;
    }

    throw new Error('Cannot move near cell: ' + JSON.stringify(cell));
}

/**
 * Move to the node in local robot map
```

```
     */
    public *moveToLocalMapNode(node: Node): Iterable<Node> {
        if (this.verbose) console.log('Moving:', node);
        while (this.yLocal !== node[0] || this.xLocal !== node[1]) {
            const { known, path } = this.localMap.bfs_path(
                this.localNode,
                node
            );

            if (path === undefined) {
                throw new RangeError('Path not found');
            }

            if (!known) {
                const known_path = this.localMap.known_path_part(path);
                const commands = this.pathToCommands(
                    known_path,
                    this.direction
                );

                if (this.verbose)
                    console.log('partial path:', JSON.stringify(known_path));

                yield* this.runCommands(commands);
            } else {
                const commands = this.pathToCommands(path, this.direction);

                if (this.verbose)
                    console.log('fin path:', JSON.stringify(path));

                yield* this.runCommands(commands);
            }
        }
    }

    public findArtag(): Command[] | undefined {
        for (let i = 0; i < 4; ++i) {
            const commands = this.getArtag();
            if (commands !== undefined) return commands;
            run_generator(this.turnLeft());
        }

        return undefined;
    }

    public abstract getArtag(): Command[] | undefined;
}
```

solution/robots/RobotReal.ts

```
import ipc_entrypoint from "../low_level_control/movement/index"
import { Robot } from "./Robot";
import { Direction, WallsState } from "../FieldGraph";
import { Command } from "../Command";
import { LLInteractionClient, create_low_level_interation_client } from
↪  "../low_level_control/low_level_interaction";

export class RobotReal extends Robot {
    private ipc_client: LLInteractionClient;
```

```
    constructor(dir: Direction, auto_scan_field: boolean = true) {
        super(dir, auto_scan_field);
        this.ipc_client = create_low_level_interation_client();
    }

    protected _forward(n: number = 1): void {
        this.ipc_client.execute_forward(n);
        console.log("RobotReal._forward(" + n.toString() + ")");
    }
    public turn(deg: number): void {
        this.ipc_client.execute_turn(deg);
        console.log("RobotReal.turn(" + deg.toString() + ")");
    }
    public getWalls(): WallsState {
        console.log("RobotReal.getWalls()..");
        const r = this.ipc_client.execute_inspect_walls()
        console.log("RobotReal.getWalls():", JSON.stringify(r));
        return r;
    }
    public getArtag(): Command[] | undefined {
        const r = this.ipc_client.execute_scan_artag();
        console.log("RobotReal.getArtag():", JSON.stringify(r.commands));
        return r.commands;
    }
}
```

solution/Command.ts

```
export const enum Command {
    None = 0,
    TurnLeft = 1,
    TurnRight = 2,
    MoveForward = 3,
}
```

solution/FieldGraph.ts

```
import { copy } from './helpers';

/** - The number of free cells at each direction:
 *     - `r_free:` right
 *     - `l_free:` left
 *     - `f_free:` forward
 *     - `b_free:` back
 *
 *  - The number of free cells before wall at each direction (if wall detected):
 *     - `r_wall`, `l_wall`, `f_wall`, `b_wall`
 */
export interface WallsState {
    // TODO: optimize localization using new walls
    // TODO: add better graph walking algorithm
    // TODO: add better localization algorithm
    r_free: number;
    l_free: number;
    f_free: number;
    b_free: number;
```

```
    r_wall?: number;
    l_wall?: number;
    f_wall?: number;
    b_wall?: number;
}

export enum CellState {
    Unknown = 0,
    Wall,
    Free,
}

/**
 * The direction of the robot:
 *   - `0`: right
 *   - `1`: up
 *   - `2`: left
 *   - `3`: bottom
 */
export enum Direction {
    Unknown = 100,
    Right = 0,
    Up = 1,
    Left = 2,
    Down = 3,
}

export function negateDirection(dir: Direction) {
    switch (dir) {
        case Direction.Unknown:
            return Direction.Unknown;
        case Direction.Right:
            return Direction.Left;
        case Direction.Left:
            return Direction.Right;
        case Direction.Up:
            return Direction.Down;
        case Direction.Down:
            return Direction.Up;
    }
}

export type Node = [number, number];

export const FIELD_SIZE = 8;

export class FieldGraph {
    public field: CellState[][] = [];

    get length(): number {
        return this.field.length;
    }

    constructor() {
        for (let i = 0; i < FIELD_SIZE * 3; ++i) {
            this.field.push([]);
            for (let j = 0; j < FIELD_SIZE * 3; ++j) {
                this.field[i].push(CellState.Unknown);
            }
        }
```

```
    }

    private *clearWallsVertical(
        [i, j]: Node,
        di_start: number,
        di_end: number
    ): Iterable<Node> {
        for (let di = di_start; di <= di_end; ++di) {
            this.updateCell(i + di, j, CellState.Free);
            yield [i + di, j];
        }
    }

    private *clearWallsHorizontal(
        [i, j]: Node,
        dj_start: number,
        dj_end: number
    ): Iterable<Node> {
        for (let dj = dj_start; dj <= dj_end; ++dj) {
            this.updateCell(i, j + dj, CellState.Free);
            yield [i, j + dj];
        }
    }

    private updateCell(i: number, j: number, val: CellState) {
        if (this.field[i][j] !== CellState.Unknown && this.field[i][j] !== val)
            throw new RangeError(`Updated scanned cell: ${[i, j]}, ${val}`);
        this.field[i][j] = val;
    }

    /**
     * Yields free nodes (for localization)
     */
    public *updateField(
        node: Node,
        walls: WallsState,
        direction: number
    ): Iterable<Node> {
        const [i, j] = node;

        switch (direction) {
            case 0:
                if (walls.l_free === 0)
                    this.updateCell(i - 1, j, CellState.Wall);
                if (walls.f_free === 0)
                    this.updateCell(i, j + 1, CellState.Wall);
                if (walls.b_free === 0)
                    this.updateCell(i, j - 1, CellState.Wall);
                if (walls.r_free === 0)
                    this.updateCell(i + 1, j, CellState.Wall);

                if (walls.l_wall !== undefined)
                    this.updateCell(i - 1 - walls.l_wall, j, CellState.Wall);
                if (walls.f_wall !== undefined)
                    this.updateCell(i, j + 1 + walls.f_wall, CellState.Wall);
                if (walls.b_wall !== undefined)
                    this.updateCell(i, j - 1 - walls.b_wall, CellState.Wall);
                if (walls.r_wall !== undefined)
                    this.updateCell(i + 1 + walls.r_wall, j, CellState.Wall);
```

```
            yield* this.clearWallsVertical(
                node,
                -walls.l_free,
                walls.r_free
            );
            yield* this.clearWallsHorizontal(
                node,
                -walls.b_free,
                walls.f_free
            );
            break;
        case 1:
            if (walls.l_free === 0)
                this.updateCell(i, j - 1, CellState.Wall);
            if (walls.f_free === 0)
                this.updateCell(i - 1, j, CellState.Wall);
            if (walls.b_free === 0)
                this.updateCell(i + 1, j, CellState.Wall);
            if (walls.r_free === 0)
                this.updateCell(i, j + 1, CellState.Wall);

            if (walls.l_wall !== undefined)
                this.updateCell(i, j - 1 - walls.l_wall, CellState.Wall);
            if (walls.f_wall !== undefined)
                this.updateCell(i - 1 - walls.f_wall, j, CellState.Wall);
            if (walls.b_wall !== undefined)
                this.updateCell(i + 1 + walls.b_wall, j, CellState.Wall);
            if (walls.r_wall !== undefined)
                this.updateCell(i, j + 1 + walls.r_wall, CellState.Wall);

            yield* this.clearWallsVertical(
                node,
                -walls.f_free,
                walls.b_free
            );
            yield* this.clearWallsHorizontal(
                node,
                -walls.l_free,
                walls.r_free
            );
            break;
        case 2:
            if (walls.l_free === 0)
                this.updateCell(i + 1, j, CellState.Wall);
            if (walls.f_free === 0)
                this.updateCell(i, j - 1, CellState.Wall);
            if (walls.b_free === 0)
                this.updateCell(i, j + 1, CellState.Wall);
            if (walls.r_free === 0)
                this.updateCell(i - 1, j, CellState.Wall);

            if (walls.l_wall !== undefined)
                this.updateCell(i + 1 + walls.l_wall, j, CellState.Wall);
            if (walls.f_wall !== undefined)
                this.updateCell(i, j - 1 - walls.f_wall, CellState.Wall);
            if (walls.b_wall !== undefined)
                this.updateCell(i, j + 1 + walls.b_wall, CellState.Wall);
            if (walls.r_wall !== undefined)
                this.updateCell(i - 1 - walls.r_wall, j, CellState.Wall);
```

```
            yield* this.clearWallsVertical(
                node,
                -walls.r_free,
                walls.l_free
            );
            yield* this.clearWallsHorizontal(
                node,
                -walls.f_free,
                walls.b_free
            );
            break;
        case 3:
            if (walls.l_free === 0)
                this.updateCell(i, j + 1, CellState.Wall);
            if (walls.f_free === 0)
                this.updateCell(i + 1, j, CellState.Wall);
            if (walls.b_free === 0)
                this.updateCell(i - 1, j, CellState.Wall);
            if (walls.r_free === 0)
                this.updateCell(i, j - 1, CellState.Wall);

            if (walls.l_wall !== undefined)
                this.updateCell(i, j + 1 + walls.l_wall, CellState.Wall);
            if (walls.f_wall !== undefined)
                this.updateCell(i + 1 + walls.f_wall, j, CellState.Wall);
            if (walls.b_wall !== undefined)
                this.updateCell(i - 1 - walls.b_wall, j, CellState.Wall);
            if (walls.r_wall !== undefined)
                this.updateCell(i, j - 1 - walls.r_wall, CellState.Wall);

            yield* this.clearWallsVertical(
                node,
                -walls.b_free,
                walls.f_free
            );
            yield* this.clearWallsHorizontal(
                node,
                -walls.r_free,
                walls.l_free
            );
            break;
    }
}

public relativePosition(of: Node, to: Node): Direction {
    const [i0, j0] = to;
    const [i1, j1] = of;

    if (i1 === i0 + 1 && j1 === j0) return Direction.Down;
    if (i1 === i0 - 1 && j1 === j0) return Direction.Up;
    if (j1 === j0 + 1 && i1 === i0) return Direction.Right;
    if (j1 === j0 - 1 && i1 === i0) return Direction.Left;

    return Direction.Unknown;
}

/**
 * Return reachable neighbor nodes
 */
public *neighbors(
```

```
        [i, j]: Node,
        allowUnknown: boolean = false
    ): Iterable<Node> {
        const allowedVals = [CellState.Free];
        if (allowUnknown) allowedVals.push(CellState.Unknown);
        yield* ([
            [i - 1, j],
            [i, j - 1],
            [i + 1, j],
            [i, j + 1],
        ] as Node[]).filter(([i_, j_]) =>
            i_ < 0 || j_ < 0 || i_ >= this.length || j_ >= this.length
                ? false
                : allowedVals.includes(this.field[i_][j_])
        );
    }

    public *dfs(
        startNode: Node,
        allowUnknown: boolean = false
    ): Iterable<Node> {
        const stack = [startNode];

        // List are just links, so we can't use list as a hashable type in JS
        const visited: { [node: string]: boolean } = {};

        while (stack.length > 0) {
            const [i, j] = stack.pop()!;
            if (`${i}:${j}` in visited) {
                continue;
            }

            yield [i, j];
            visited[`${i}:${j}`] = true;

            stack.push(...this.neighbors([i, j], allowUnknown));
        }
    }

    private _bfs_path(
        startNode: Node,
        endNode: Node,
        allowUnknown: boolean = false
    ): Node[] | undefined {
        const q = [[startNode]];
        const visited: { [node: string]: boolean } = {};

        while (q.length > 0) {
            const path = q.shift()!;
            const [i, j] = path[path.length - 1];

            if (`${i}:${j}` in visited) continue;
            visited[`${i}:${j}`] = true;

            if (i === endNode[0] && j === endNode[1]) {
                return path;
            }

            q.push(
                ...[...this.neighbors([i, j], allowUnknown)].map((neighbor) =>
```

```
                    copy(path).concat([neighbor])
                )
            );
        }

        return undefined;
    }

    public known_path_part(path: Node[]): Node[] {
        const ret: Node[] = [];
        for (const [i, j] of path) {
            if (this.field[i][j] !== CellState.Unknown) {
                ret.push([i, j]);
            } else {
                break;
            }
        }

        return ret;
    }

    public bfs_path(
        startNode: Node,
        endNode: Node
    ): { path: Node[] | undefined; known: boolean } {
        const p1 = this._bfs_path(startNode, endNode, false);
        if (p1 !== undefined) return { path: p1, known: true };

        return {
            path: this._bfs_path(startNode, endNode, true),
            known: false,
        };
    }

    public toString(): string {
        return this.field.map((line) => line.join(', ')).join('\n');
    }
}
```