AURO - Report

Y3889883

I. Design

My overall algorithm for my robots is fairly simple; be greedy. I chose this due to the nature of the problem and in particular the swapping mechanics. The items can be swapped between clusters and will continue to spawn into that cluster, meaning that, one could swap more value items into more favourable positions allowing for quicker collection. For each robot, the algorithm works as follows: find the nearest item and pick it up, then, if it can find any item of higher value, pick it up. This step is repeated until it can't find any item of higher value, in which case it will go home. The change in how it is greedy is important. First looking for the nearest and then a higher value. This results in any lower value items that are in close favourable positions often getting swapped with a higher value item.

This approach has three flaws that I can think of. The first flaw is: that it can take a while for the higher-value items to get into more favourable positions. It would probably be more efficient in the long term to swap all the higher value items into more favourable positions before collecting any to home however I believe my solution has a good balance of short-term and long-term thinking enabling it to perform well in both scenarios. There is also something to be said about having a simpler solution. The less complexity the easier it is to change the implementation without causing bugs. The second flaw is: that if a robot has a lower-value item but can see a higher-value item that is in a more favourable position, it will still swap it, putting the higher-value item in a worse position. This is quite a major flaw and could be remedied by having a heuristic measure for how favourable the position is and comparing the positions of the two items' original positions. However, implementing this fix would have been challenging, especially given the inaccuracies in item position estimation. Moreover, this situation is relatively rare, as robots are more likely to find items in more favourable positions. The third flaw is: that having lots of robots can often result in them targeting the same or close items. I resolved this with strict management of the robots trying to make sure that they don't target the same items or get close to each other.

If this approach was all that was needed for the robots to work. My implementation would have been a lot smaller and easier but when working with multiple robots, collision avoidance becomes a much bigger issue. This covers around half my implementation so is important to design well. There are three main rules I designed to help my robots not collide with each other: The robots must never target a similar position, when going to the starting area they must go to one of the starting positions (where each robot spawned), when getting too close to each other, they must go to a starting position. Theoretically, only the third rule is necessary as moving away to a clear location should always stop collisions. However, this is very time-consuming so the other rules were introduced to make sure the location is safe and to avoid situations where the robots do get too close.

Separation of responsibility is always a difficult task and this is especially true for this problem. I have organised my solution into three components: The item filter, the robot coordinator and the robot controller. The Item filter takes the messages from the item sensor and filters out any items that the robot shouldn't be interested in such as what other robots are carrying and keeps the ones that the robot should be interested in, the nearest of each different value. The robot coordinator takes in information about each robot and outputs information about the other robots, what places are available and if robots are too close to each other where they should go to get away from each other. The robot controller, you guessed it, controls the robot. It takes in the information from the item filter and the coordinator but also the odometry and item holder information. With this information, it decides where to go and tells the basic navigator from nav2 to go there. All of this could have been done just in the robot controller but this would lead to hard to maintain and understand mess. I separated it the way I did because it makes sense to have the robots publish and subscribe to a stand-alone node rather than to each other. This allows it to coordinate better as it has all the information, doesn't have a bias and takes some of the workload of the robot controllers. The amount of navigation logic to put in there was a difficult decision as I wanted to it have the role of giving guidance and information rather than making the decision for the robot. This is something I didn't completely succeed at as the coordinator does say where to move to in the case where robots are too close to each other.

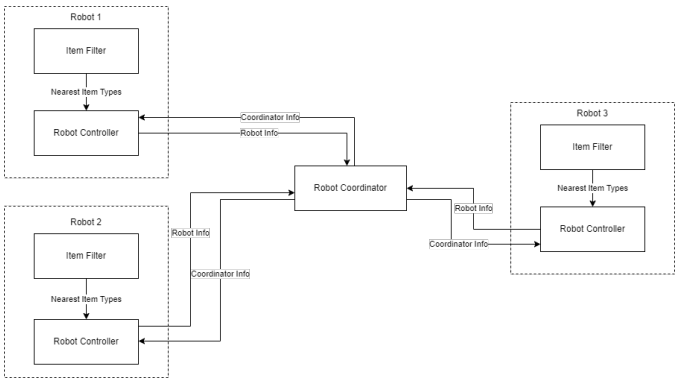

Fig. 1. Block diagram of the solutions ROS nodes and the messages sent between them

II. Implementation

I have implemented my solution such that the robots don't really set their state but rather their state is defined by what they can see and what it is currently holding. After computing this state, it then computes a target navigation goal. If that place is already being targeted it will recompute a new state and target until it finds one that isn't being targeted something that is guaranteed to happen as it can always go to a starting position.

Computing the state is pretty simple. It just follows some conditions according to the table shown in Fig. 2:

		Holding			
		None	Red	Green	Blue
Seeing	None	Continue	Continue	Continue	Go Home
	Red	Find Nearest	Continue	Continue	Go Home
	Green	Find Nearest	Find Green	Continue	Go Home
	Blue	Find Nearest	Find Blue	Find Blue	Go Home

Fig. 2. Logic to decide what state the the robot is in.

Note that there is a priority to the states that isn't stated in the table in case it tries to target something that's already being targeted. Computing the target is a little more complex. There are three different things it could be targeting: the same thing it's already targeting, an item or a starting position. Out of these, only the item is a little complex. It uses the item's diameter to estimate a distance and the item x value to estimate an angle which it can then use to compute the coordinate relative to the robot. This was made complex due to the camera not being directly centred on the robot meaning that: certain bearings would look different depending on the angle you look at it with but more importantly the item's x value compared to the angle of the item will change depending on distance.

The coordination between robots was probably the hardest to implement. I started with no robot targeting the same position. This is done by all robots publishing what position they are targeting and the coordinator collecting them all and republishing them all with the respective robot ID. Each robot can then individually figure out if a robot is already targeting the area they were about to target and then choose a different target. For this problem, this is very inefficient. It would be better if the robots collected each other's targets directly, as there would be fewer messages and also wouldn't need to wait for the coordinator. I decided against doing this as I knew I would need the targeting information for the other rules and the time difference isn't large enough to make a great difference. For the starting position rule the robots all publish their starting pose. The coordinator subscribes to this and develops a list of all starting positions. It then filters these based on whether any robot is targeting a starting position to publish the list of available starting positions that a robot can choose from. This gives the robots more options rather than requiring them to go back to their own starting position. This is more suited to the coordinator than having it in the robot controller as it is more of a centralised list that all robots must follow even if they could work it out by themselves. The final rule is about when they get too close to each other works by each robot sending updating the coordinator with their positions. The coordinator subscribes to this and maintains a list of their positions. It then goes through the list and if any are too near to each other it computes the near available starting position that is nearest to the opposite bearing to the average bearing of the near robots and publishes the position the robot should go to. This is the most suited to the coordinator as if each robot did it individually they could end up targeting the same starting position before the available list of starting positions is updated. I didn't use any actions in my implementation, instead only using topics to communicate across nodes, as I don't believe any of the communication quite fit the use of actions. The available starting positions had potential as robots could have used requested the most up-to-date list but that doesn't quite fit the use case for actions. The most suitable would probably be going to a starting position after getting too near another robot. As it could be used to make sure they do move away from each other. If I had decided to have all the decision and targeting logic in the coordinator instead of the controller there definitely would be a use case for actions to keep track of the robots as they move towards their targets.

The robots need to know their ID so that they can send it along with the messages they send so that they know which public messages are addressed to them and so they can tell which item they are holding. I gave them this information through the use of parameters so that each automatically knows. This also allows the robot controller to use the same callback for each robot as it can distinguish between robots based on their ID and then add it back when sending a message to that robot.

III. Analysis

For my analysis firstly, I watched the simulation. This was to see if the robots did what I expected them to do. Make sure that: the robots make the decisions they are supposed to (described in Implementation) and result in the behaviour I expect (described in Design). The result of this form of analysis is that my robots perform pretty well. They make the right decisions almost all of the time which results in the correct outcome almost all of the time. This however means there are cases in which they don't make the correct decision due to a bug in my implementation. The only cases I saw of this is where sometimes when robots get too close to each other, they are meant to target different homes but sometimes they would target the same home. This isn't doesn't have a big impact on the effectiveness of the solution as one of them always targets a different home soon after. There is another case where robots can sometimes target items that are too close together at the start before the coordinator has a chance to publish already targeted locations. This is more impactful as it can cause robots to get too close and therefore have to go to a starting position. There is also a case where the robots make the right decision according to my design but it results in the wrong behaviour. Sometimes when robots get too near each other they can still run each other even if they targeting safe spaces. I think the blame for this is on nav2 however I can by no means be certain of this and the cause of this may lay on my shoulders. The other form of analysis I did

was with collection logs. I did a run for 10 minutes on 10 different seeds (bear in mind this was around 3-4 minutes of sim time) to find quantitative data on how well my solution performs. The first analysis I did was finding how consistency of my solution.

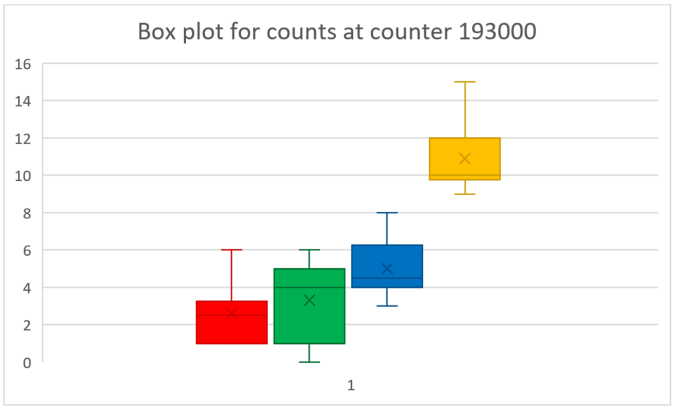

Fig. 3. Box Plots for amount of items collects at counter number 193000. Yellow is total, other colour match the corresponding item

Fig. 3 shows box plots for the count of items for each colour and the total at the earliest time one of the simulations ended. This shows that my solution is very consistent at collecting the highest value items and pretty consistent in the total amount collected, however, this does not show if my solution is consistent in the value it collects. The second metric I used is the mean average for my solution.

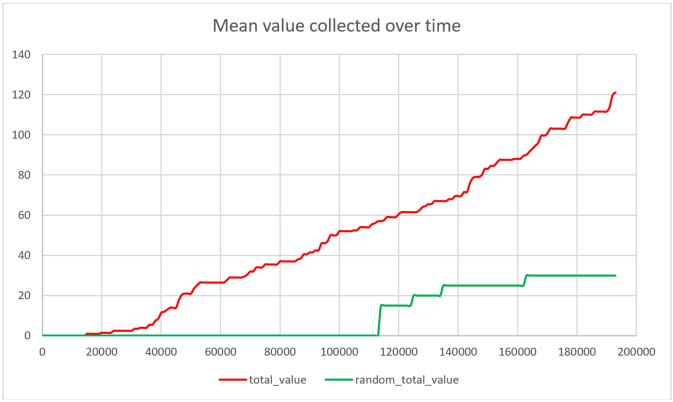

Fig. 4. Mean value of my solution vs a random walk. Red is my solution. Green is the random walk

Fig. 4 shows the average total value of my solution compared to the average total value of the random walk solution. This shows that my solution outperforms the random walk at all stages of the simulation.

Fig. 5 shows the average value collected over time. This shows that I get much more value from the blue then the other colours suggesting that my design works as intended and suggests that my design is good for collecting value.

Fig. 6 shows the average blue count over time with a generated polynomial trendline. This clearly shows that the rate of blue item collection increases over time making it clear that my design has the intended result and also pretty works

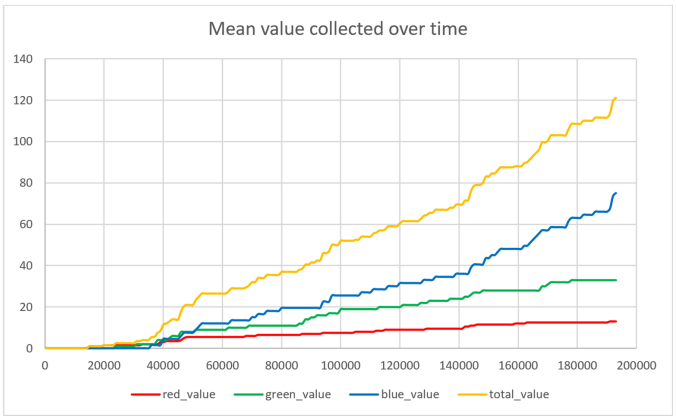

Fig. 5. Mean value of items collected by the robots against time

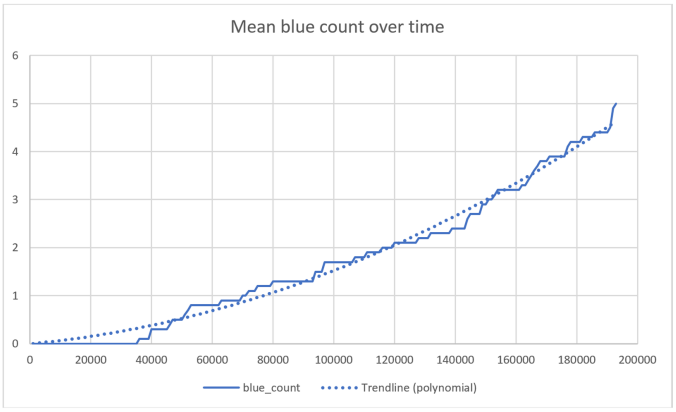

Fig. 6. Mean amount of blue items collected against time with a polynomial trendline generated by excel

very effectively. The trendline does start at 0 when the earliest any object is collected is around 14000 so it might have been fairer if the trendline starts there instead, I would have done this but the program I used to create the polynomial doesn't have this feature as far as I'm aware.

I chose to use the mean for the graphs. I would have preferred to use the median as they are less affected by outliers, but because the dataset was small, and the simulations were not run for very long, the jumps were much larger making it harder to interpret meaning or trends from the graphs.

Bear in mind that these graphs are limited by the size of the dataset. If I had more time I would run the simulations for longer, as I do not believe my solution had reached its maximum effectiveness. I would also run the random walk over all the random seeds rather than just the one to make sure that the one it was run on wasn't an outlier.

IV. Evaluation

A strength of my solution is that the speed at which 'value' is collected increases with time while having a decent collection rate at the start as well. It outperforms the turtlebot3_gazebo.turtlebot3_drive solution by a lot although this isn't much of a feat.

The analysis shows that my design is effective at achieving the goals it sets out to achieve: It performs well at all stages,

its effectiveness increases over time and it is simple enough to easily understand. My design also handles multiple robots well and coordinates them a suitable amount, avoiding crashes and inefficiencies most of the time. If I were to design this again, the only things I might change are making the robot avoid items when going to a starting position as this would increase the value collected at that time and if the robot already has a high-value item then it has likely been swapped. However, this would not be a perfect solution as it may be missing out on swapping it into a more favourable position. I might also change how much the coordinator interacts with the robots, maybe being more forceful and changing routes to avoid robots getting in each other's paths.

My implementation has some real strengths such as the item estimation being accurate in the ways that matter and the state decision making. However, it does have some major flaws, crashing at all is very problematic as this completely ruins the navigation, basically making the robot useless for the rest of the run. As crashes at some point are inevitable given enough time for most solutions, I don't think the crashes are the weakness. I think the inability to recover from crashes is the greatest weakness of my solution and if I were to reimplement my solution, this would be something that I would attempt to change.

In a real-world scenario, my solution would perform terribly. The main reason is that it uses odometry to get its location and bearing. This only works in simulation as there is no wheel slipping which would result in odometry drift. In reality, the position estimate would slowly become worse and worse over time until the robot thinks it's in a place that it's not. This could be improved by using a different localisation technique such as using the location nav2 thinks the robot is. In reality, it is also not guaranteed that you have a map of the area. This would impact the navigation of the robot somewhat, although nav2 should be able to deal with that problem. Another potential problem is the gradient of the ground. In simulation, the robots have only been tested on a flat surface. The filtering out of items that a different robot is carrying might work against us if the gradient changes as items may appear above the robot. The robots also might behave poorly on gradient e.g. failing to stop in time. You might be able to account for a static gradient if you can detect it but one that changes could mess with the positions of the items too much. The simulation camera acts differently from real world lens. It has no lens distortion which is why I could estimate the angle of the items in regards to the robot so accurately. This would break if using a real-world lens and having a decent location estimate for the items is the basis of my solution. With enough maths, this can be accounted for. An image can be effectively flattened if the distortion of the lens is known.

V. Safety and Ethics

Ethics can be interesting as everyone has a different view on what being ethical looks like. I am writing mainly from the UK RAS Network's view on ethics. In regards to item retrieval by autonomous robots I think there could be 5 main potential ethical issues. The first issue is employment. Item collection by autonomous robots already exists and has already replaced human workers by companies such as amazon. This is a big issue and can lead to disastrous consequences for those it replaces. The second is bias. One way this could come into play is if a robot is retrieving items and bringing them to a human, but if the human detection system has been trained on an ethnically biased set of images then it may fail to recognise certain groups. This could actually be an issue for my solution as even though it has no human detection, if it does replace any jobs, it will probably have a bias for which jobs it replaces. The third is opacity. This tends to relate to more dubious decisions that could be unjust but some people view any intentional block on information unethical. In regards to my solution, the code is private and only viewable to me and the people involved in assessing it. However, if it were to be used in a real-world scenario I would hope that it would be publicly viewable, especially as it isn't exactly ground-breaking. The fourth is oversight. An overseer should be able to manage an autonomous robot and enact some level of control over it so that the robot doesn't do something bad. In my solution are some options to enact control. You can teleoperate the robots however this is flawed as you will have to 'fight' for control with the nav2 also sending movement messages. Unfortunately, the only other way to enact any control is by shutting it down. The fifth issue is safety. This is an issue I think the vast majority of people agree is important, however how important still depends on the person.

Safety for autonomous robots is difficult. But for an item collection robot it may not be too hard. Risk is probability times severity and is generally thing that people want to minimise. An item collection robot may not have to interact with humans. As (most) people view damage to humans as much more severe than damage to assets, the risk for robot that doesn't have to interact with human is dramatically lowered. If a robot does have to interact with humans some issue a designer might have to consider are distinguishing between robots and humans, or how to get close to a human (to hand them an item) without crashing into them. It is important to bear in mind that it is impossible to eliminate all risk and that some risk is acceptable to everyone as almost every action someone does involves some amount of risk. It is also important to note that you shouldn't always strive to minimise the risk the minimum amount at the cost of something else (such as efficiently or cost). If people were to do this in their everyday life, cars would not exist.

If my solution was to be deployed in the real world I think the risk would be fairly low, even if it had to interact with humans. This is mainly due to the severity being quite low, if a robot crashes into a human, it is going to travelling pretty slow and due to its size, it can't inflict much damage. However, the probability of an incident is pretty high as even in a simulation environment the robots sometimes crash. If it were to be deployed in the real world some things I could have done to make it safer, include: making the object avoidance stricter, getting rid of the recursion, writing it lower level language to have finer control over things like memory. This way I could avoid allocating to a heap of unknown size.