EGH446 Autonomous Systems

**Major Project – Individual Report.**

Instructions: replace the yellow highlighted text with your own words (that is, delete the yellow text). You must use this template (not alternative template).

Individual details

|  |  |
| --- | --- |
| Author | Cody Cripps |
| Sub-systems | My Subsystem:   * Receding virtual waypoint system   Other:   * Heading and Velocity Control Systems * Diagnostic system |
| Project Partner | Pierce Braithwaite |

System description (group’s own words, allowed to be same as your group partner)

|  |  |
| --- | --- |
| Overview of the system  (Cody) | The system detailed in this report is a robot guidance and control system, capable of taking some list of arbitrary waypoints and navigating a differential drive robot between them. The system developed was found to be effective both at reducing error in path tracking and at quickly and efficiently completing set paths.  The individual required functionalities of the system were isolated as separate subsystems. These were as follows:   * Heading Control Subsystem * Velocity Control Subsystem * Receding Virtual Waypoint (RVWP) Logic Subsystem.   Alongside these systems a diagnostics system was also run to evaluate the system.  These were connected as can be seen in Figure X.    Figure 1: Simulink diagram of the full system.  These interconnections of the subsystems will be further explored in the section titled interfaces  Each of these subsystems was implemented and tuned to optimise runtime while minimising cross-track error from the ideal path.  Heading Control Description  The heading control system’s task is to adjust the robot heading so as to best reach the inputted goal point. The subsystem took the difference between the robot’s heading and the angle from the robot to some inputted X and Y position, as the error argument. This error was then minimized using a PID controller.  A PID controller was chosen for this application as the system was found to be prone to oscillation across the ideal path, as well as some maintained set point error.  The output of this system was a turn rate which would be fed into the vehicle dynamics.  Velocity Control Description  The goal of the velocity control system is to maintain the balance between required precision and the speed at which the system would complete its task. This means that when the robot is navigating to a distant point the robot would not require high precision and can therefore be given more speed. Conversely when the robot is navigating to some close position it may need to make more precise movements and therefore the forward velocity would need to be smaller. This was achieved in practice using a simple proportional controller which took the distance to the control point as the error value and outputted a desired velocity input to the vehicle dynamics.  Receding Virtual Waypoint Guidance Subsystem  This system handled the robot's guidance using Receding Virtual Waypoint (RVWP). The virtual waypoint is defined in this implementation as a point on the ideal path some specified lookahead distance from the robot. This look ahead distance is a configurable parameter. The goal of the RVWP guidance system is to effectively maintain the ideal path rather than navigating directly to the goal. In arbitrary examples this may not seem necessary, however, in real world, cluttered environments it is far more like that the ideal path was designated to avoid obstacles and is therefore just as relevant as the waypoints themselves.  Diagnostics Subsystem  The task of the diagnostic subsystem is to monitor the performance of the system and to end the simulation when the final goal is reached. The monitoring of the system’s performance is achieved by calculating the cross-track error of the system. This error represents the distance that the route of the robot has deviated from the planned path and is the best indicator of the system’s performance. This error signal is then outputted and sent to the simout so that it can be analysed to find the maximum and average error. It is also outputted to a scope so that the error can be viewed graphically. The control function also outputs a Boolean value that returns true when the final goal is reached. When this value reads true the simulation is stopped.  Subsystem Interactions  Overall, these systems interacted to produce the desired output. The RVWP system would output an X and Y point that the heading and velocity controllers would convert into vehicle readable desired turn rates and velocities.  When the velocity control was given too high of a gain, the effect of the heading control was negligible. A similar outcome would occur if the velocity control was allowed to vary to 0 and the heading control gains were too high, in this case the robot would spin on the spot. In general, the velocity control was the touchiest system when given too much power (a higher gain) and therefore contributed less to the overall control performance than the heading controller. |
| Performance | 1041.2 seconds |
| Interfaces  (Cody) | Figure 2: System Diagram  Figure 2 displays the system diagram for the interactions between subsystems. The Diagnostic system outputs only to the user, taking input from the RVWP system. The RVWP system outputs the cross track error to the diagnostics, as well as a stop signal when the final waypoint is reached. It takes pose input from the Vehicle Dynamics. The control subsystems (Heading and Velocity) take a x and y goal point as input from the RVWP subsystem. The Heading control converts this goal point into a desired turn rate for the vehicle dynamics while the velocity control converts into a velocity input. |
| Limitations  (Pierce) | During testing of the system, the simulation never resulted in a failure case. However, there were some limitations to the system that were observed.  The main limitation of the controller is that it is most suited to larger test tracks such as the assessed waypoints and the spanner track. The constant values implemented in the PID controller are specifically tuned for peak performance in larger tracks. When completing a larger track, the system may experience a larger error signal, which requires a larger amount of control. This results in much sharper turning and overcorrection, resulting in more deviation from the planned path.  The distance threshold for the robot to start moving towards the next waypoint as well as the look ahead distance of the receding virtual waypoints were 0.2 metres and 0.1 metres respectively. On larger test tracks, these distances are quite small and produce negligible error. However, on smaller tracks these distances are larger compared to the scale of the track, which leads to a comparatively larger error. On much smaller tracks where there are straights of 0.4 metres the robot is barely able to reach the path before it begins to turn towards the next waypoint. The velocity control is also tuned to be effective for larger tracks. Given that the smaller tracks require more precision in order to navigate them accurately, the larger speed limits of the system are not suitable. As a result, the system experiences larger cross track error and more overshoot when changing direction.  Another limitation is that on smaller tracks the controller doesn’t have enough time or distance to fully correct the heading of the robot before reaching the waypoint. This, along with the overshoot caused by higher velocity, leads to a much less accurate route travelled by the robot.  When changing direction, the robot would sometimes do a full 360 degree turn before turning to the correct heading. This causes more cross-track error and slightly delays the system. While the results are negligible in larger simulations such as the assessed waypoints and the spanner track, it makes a much more noticeable difference in smaller tracks, such as the CIRCLE\_HOME track as the circles created are comparatively larger to the scale of the track.  Another limitation that occurs in the system is moving the robot to the starting point at the beginning of the simulation. When the simulation begins the robot appears at point (0,0) before immediately moving to the starting point. This causes a straight line from point (0,0) to the starting point on the figure representing the route of the robot, as the simulator sees this as part of the robot’s movement. This also causes an initial spike in the error signal, as the system calculates the cross-track error for point (0,0). This affects the mean of the error signal as well as the angular velocity. The integral and derivative controllers use previous error values to calculate the angular velocity. This causes an initial error in the heading that is quickly fixed by the controller. |
| Figure of the route defined by the waypoints and vehicles path. |  |
| Report on cross track error  (Pierce) | The absolute cross-track error of the system while completing the above route can be observed in the figure below:    The error signal has an initial spike of 20 as the simulator moves the robot to its initial position. Excluding the spike, the error remains close to zero for the whole simulation. As can be seen, there are small spikes in the cross-track error throughout. These spikes occur when the robot reaches a waypoint and changes direction. The mean of the error is 0.023 metres excluding the initial spike and including the spike the mean climbs to 0.031. Not including the spike, the maximum cross-track error that occurs is 0.1857 metres. This occurs when the robot is changing direction at the seventh waypoint. The robot makes a full 360° turn before turning to the correct heading.  When completing a step function with a way point of x = 10, y = 0 and an initial heading of 30° the system produced the following cross-track error:    The peak error experienced by the system is 0.0565 metres and it takes the system 1.64 seconds to reach zero error. However, the error began to stabilise after approximately 3.5-4 seconds. The mean error is 0.023 metres. There appears to be no steady-state error and any present could be considered negligible. |

Sub-system(s) description (individual student’s own words)

|  |  |
| --- | --- |
| Overview of function | The contribution detailed in this report is the receding virtual waypoint system. This subsystem handles the guidance of the robot, aiming to track the ideal path between the given waypoints. In this implementation the virtual waypoint was defined as a point on the ideal (straight) path between the previous and next waypoints, which is some lookahead distance from the robot’s parallel point on the ideal path.  This is done so that the system could more effectively track the ideal path or regain this path if/when it is lost. In more arbitrary structured environments, without obstacles, it would be more efficient to use control to direct the robot along the shortest route from it to the next waypoint. Though this may be true in those environments, given a real world environment is cluttered and unstructured a robot guidance system should aim to follow the calculated path, which in this case is the path between the waypoints, as this was likely calculated to avoid obstacles.  This system also calculates when to switch to the next waypoint and when the entire path is completed.  To calculate virtual waypoints this system requires the position of the robot as well as the previous and next waypoints. The system does not consider the angle of the robot. This is as the robots current angle does not effect the intended path in any relevant way. If obstacle avoidance were to be considered this may need to function differently, depending on proximity to obstacles.  Calculations    Figure 3: Geometry for calculating virtual waypoint  The goal point was calculated geometrically. These calculations are fairly simple, just calculating up the path from the previous waypoint using trigonometry.  First the angles of the path () and the robot () were calculated:  Following the intermediate distances were calculates, that is the distance from the previous waypoint (dprev­­­) and the cross track distance (dpath):  Following this the distance along the path to the goal/Virtual Waypoint, was calculated:  With these intermediate calculations we can calculate the x and y value of our virtual waypoint as below:  Other system functionality  This system also calculates the cross track error for diagnostics, which is taken from the intermediate steps above. It is also where waypoint handling is completed. Switching to the next waypoint when the robot has come within a reasonable threshold of the current waypoint and stopping running the simulation when the full track is completed. |
| Performance analysis | This system can easily be isolated and tested for varying cases by calling the calculations from a test script and plotting the result. Test cases were as follows.   * Positive path angle   + Varying distances, left and right of the path * Negative path angle   + Varying distances, left and right of the path * Zero path angle   + Testing at varying y angles, maintain x. This is to test that in the simplest case the virtual waypoint will be in the correct place.   Test 1: Positive Path angle  X0 = 2;  Y0 = 2;  X1 = 4;  Y1 = 4;  d = 1;  With a lookahead distance of 0.5 and possible robot points of (3,3.5), (3,2.5), (2.5,4), and (2.5,1.5) the system results with the following Virtual Waypoint output:    Figure 4: RVWP Test with a positive path angle  Test 2: Negative Path angle  X0 = 2;  Y0 = 4;  X1 = 4;  Y1 = 2;  d = 1;  With a lookahead distance of 0.5 and possible robot points of (2.5,3), (3,3.5), (2.5,2), and (3.5,3.5) the system results with the following Virtual Waypoint output:    Figure 5: RVWP Test with a negative path angle  Test 3: Zero Path angle  X0 = 2;  Y0 = 2;  X1 = 4;  Y1 = 2;  d = 0.5;  With a lookahead distance of 0.5 and tested points at intervals of 0.5 between y = 3 and y = 1, at a constant x = 3. the system results with the following Virtual Waypoint output:    Figure 6: RVWP Test with a zero path angle  Test 3: 90-degree Path angle  X0 = 2;  Y0 = 2;  X1 = 2;  Y1 = 4;  d = 0.5;  With a lookahead distance of 0.5 and tested points at intervals of 0.5 between x = 3 and x = 1, at a constant y = 3. the system results with the following Virtual Waypoint output:    Figure 7: RVWP Test with a 90 degree path angle  Discussion  The system performed as expected in all cases, with points on the same line perpendicular to the path outputting the same virtual waypoint and all robot points also outputting virtual waypoints that appear the correct distance along the track. In the 90 degree and zero angle tests it can be observed that the virtual waypoints are in fact the intended 0.5 lookahead distance from the parallel point on the ideal path. In the 90 degree test this is shown as the virtual waypoints occurring at y = 3.5, where the robot y was a constant 3. Similarly in the zero angle test the waypoints were at x = 3.5 where the robots where at a constant x = 3.  In all cases the virtual waypoint was forward along the ideal path which is the desired behavior. |
| Pros | The receding virtual waypoint (RVWP) system, via geometric calculations was found to be the most effective for the application, when compared to the other guidance systems considered. Before considering other guidance methods, the need for a guidance system was deliberated. The other methods considered were receding virtual waypoints calculated with a simultaneous equation solver, the use of a line follow algorithm rather than a simple move to point controller. A simpler intermediate waypoint system was also considered, which would calculate a number of waypoints depending on the distance between major waypoints prior to runtime.  When considering not including any guidance logic the RVWP system was clearly more efficient. The RVWP system allows the robot to track the ideal path between points, which is generally more efficient. It can also be seen that this lowers the chance of the robot straying into objects around an unstructured environment.  In comparison to the simultaneous solver RVWP method the geometric calculations were considerably less computationally intensive. This is as the simultaneous solvers in matlab use complex symbolic methods which take up a lot of resources. On the mid-level computer system used for testing this resulted in an extremely slow simulation response of less than 0.25 seconds simulation time per second real time. The code for the geometric method is also more readable as you can step through the process rather than just looking at a couple equations.  When compared to the line follow algorithm performance differences were minimal. This is as the effective outcome of incorporating the guidance into the control law is the same as directing to a virtual waypoint for the most part. The difference comes when looking to expand the systems functionality. Their could be situations were a disturbance may put an obstacle in the path back to the desired line, if the robot were effected by an external force to move it too far off track for example. In this case obstacle avoidance could not be easily implemented within the control law. However, with the RVWP system one would be able implement a system to augment the virtual waypoint system with some avoidance logic, such as Artificial Potential fields.  The logic behind the intermediate waypoint system was that this would cause the position control to direct the robot towards the track in a direct route and be simple to implement. This was however found to cause inefficiency in the robot’s movement system. The RVWP system however allowed the robot to track the line efficiently and handle disturbances quickly. It can also be seen that precalculated intermediate waypoints cannot be adjust for disturbances, therefore allowing the possibility of the robot moving past and having to track back to the last intermediate waypoint, effectively working counter to the intended direction. This cannot, however, occur with RVWP as the virtual waypoint is calculated in ‘Real-Time’ as an effective waypoint to move towards to regain the ideal path and move towards the next major waypoint. |
| Cons | Despite finding it to be the most effective method, the Geometric RVWP system did have some negative attributes when compared to the aforementioned alternatives. As in the previous section the other methods that were considered were as follows:   * No Guidance Logic * An RVWP system that used simultaneous equations to find a solution. * An algorithm that incorporated the path guidance into the control law (control along a line). * A simple precalculated intermediate waypoint system.   The only con to using RVWP over not using any guidance is that in cases without obstacles, the original ideal path may not always continue to be that. For example, if a disturbance puts the robot off of this path in an obstacle free environment, the most efficient path becomes the direct route to the next waypoint. In this case the RVWP will actually slow down the robot’s activity by forcing it to move back to an arbitrary path that is no longer relevant.  In using simultaneous solutions for RVWP two possible points on the track will be automatically calculated, from these the logic can select the ideal point to track to. The geometric method only looks forward along the path and therefore cannot be used to consider alternate virtual waypoints without adding a secondary calculation and expanding the system.  Using a line follow algorithm would have resulted in a more compact solution. Wherein the RVWP solution required the lookahead distance and the control parameter to be tuned somewhat separately and could result in an extra layer of complexity in design, the line follow algorithm would have one set of parameters to be used to tune both the control and guidance.  When considering the intermediate waypoint system, it can be seen that the RVWP system would be more computationally intensive during runtime. This is as it would be continuously calculating virtual waypoint with the changing position of the robot, where the intermediate waypoint system calculates its waypoints prior to runtime.  It can also be seen that in this version of the system there is no way to account for the robots existing momentum or heading. This could cause the robot to run into obstacles or simply track an inefficient path if not considered. |
| Limitations | The RVWP system was found to provide efficient path tracking in most realistic cases. Despite this some limitations can be recognized, especially in exceptional circumstances.  The system cannot handle significant overshoot of the major waypoints as it views the paths between waypoints as infinite lines, not line segments. Given a situation where the robot is some non trivial distance from the ideal path and the system calculates a virtual waypoint beyond the major waypoint, a failure case would occur if the robot tracking this virtual waypoint doesn’t come within the threshold distance of the actual waypoint. This failure case would cause the robot to continually track in the given direction.  This could be fixed by calculating both possible virtual waypoints (the one forward in the path and the one backwards, at the chosen lookahead distance), and selecting the virtual waypoint closest to the goal point.  Another limitation in this system that it only gives the control system the virtual waypoint. This means that the control system cannot be optimized to run faster when on the ideal path. |

Description of all code or Simulink blocks developed by student. (individual student’s own words)

|  |  |
| --- | --- |
| RVWP\_find | Purpose:  This function calculates the virtual waypoint on the ideal path based on the robots position and the location the previous and next waypoints.  Inputs:   * robotx – The x coordinate of the robot * roboty – The y coordinate of the robot * X0 – The x coordinate of the previous waypoint * Y0 – The y coordinate of the previous waypoint * X1 – The x coordinate of the next waypoint * Y1 – The y coordinate of the next waypoint * d – The lookahead distance   Outputs:   * solx – The x coordinate of the calculated virtual waypoint * soly – The y coordinate of the calculated virtual waypoint   Functions this function/module called by.  This Function is called by the RVWP module |
| RVWP\_find | Purpose:  This function handles the RVWP system, performing the waypoint handling and outputting virtual waypoints to the control system. This includes calculating cross-track error and switching waypoints when required.  Inputs:   * state – The robot’s current pose * Xd – the X coordinates of all waypoints * Yd – The Y coordinates of all waypoints * d – The lookahead distance   Outputs:   * X – The x coordinate of the calculated virtual waypoint * Y – The y coordinate of the calculated virtual waypoint * d\_error – The instantaneous cross track error * stop – Boolean signal to stop processing when the final waypoint is reached.   Functions this function/module Calls.   * RVWP\_find |