Group B261 Programming mini project

❖ 20153397 - Kasper Tandrup

❖ 20153630 - Laura Montesdeoca Fenoy

❖ 20153886 - Nicolaj Møller Kristensen

❖ 20154124 - Torben Moesgaard Nielsen

❖ 20154272 - Michael Falk Vedel

❖ 20155464 - Christian Frøkjær

❖ 20155611 - Frederik Marc Luther

Introduction

For the programming mini project, we have decided to go with the software developed for our P1 project. Specifically we will in this project focus on the UI, what lies underneath it and how the movement works. We will also provide a description of how these nodes work together. The source code for this project is available on github[1].

Program Requirements

For our mini project software, is to work properly we need the three nodes created in P1 and the turtlebot hardware. The software should be able to work on other platforms as well, as long as these supports the ROS platform and have a screen and keyboard for the UI.

At bootup of the system, several nodes are needed to be launched, first of we need a core, for this 'roslaunch turtlebot_bringup minimal.launch' will be used. When the robot has been brought up it has to load a premade map we will use ´roslaunch turtlebot_navigation amcl_demo.launch map_file:=/home/turtlebot/maps/test5.yaml´, from which it gets the coordinates to go to. For good measure rviz should also be launched and told, where the robot approximately are on the map. Lastly we have to run our nodes, called 'ear1', 'ear2' and 'mouth'.

The program

In this mini project our software will have an alarm, that will be a UI where the user can set an appointment, based on the time they want it to occur and the location in which they want the robot to move to. These are preset coordinates based upon a map.

When the alarm triggers the schedule node will then notify one of two nodes with different coordinates. This will be done using publisher/subscriber, to communicate between the nodes.

[1]https://github.com/kapper4444/Miniproject.git

Startup

When the program starts up, it loads five libraries. The first one is ‘iostream’ which is a standard library used in c++, it contains all the basic functions such as ‘cout’, ‘for’, ‘if’, 'while' etc.

The second is 'iomanip', where we use the function 'setw' which is a function that sets the width for output operations (this function will be explained in more detail in a later section).

The third library is 'ctime', this library contains functions which allows the user to work with time (for the program).

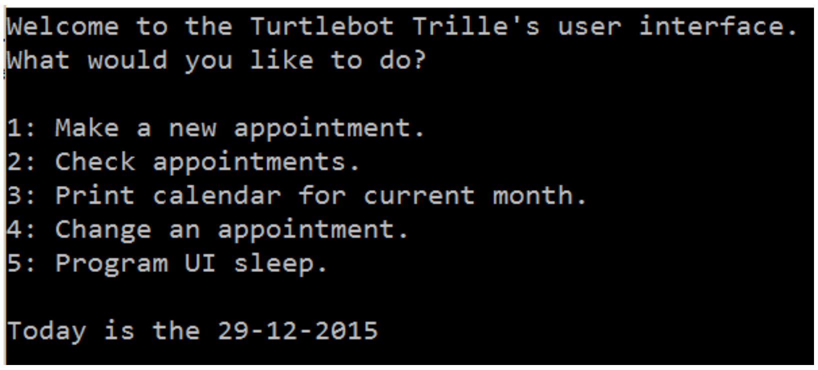
The fourth is 'cmath' which contains mathematical functions which allows the use of math expressions such as the trigonometry functions 'sin', 'cos' and 'tan'.

The fifth is 'vector', this is a library for the vector function and it contains functions, which allows the user to manipulate the vector (this will be explained in a later section).

After having loaded these libraries, it then declares some functions, voids and global variables and finishes the startup by printing the menu.

In the menu, the user now has a choice between five operations, which can be seen in figure 1.

This choice is made with the 'if' statement.



```
Welcome to the Turtlebot Trille's user interface.
What would you like to do?

1: Make a new appointment.
2: Check appointments.
3: Print calendar for current month.
4: Change an appointment.
5: Program UI sleep.

Today is the 29-12-2015
```

Figure 1 The menu

Depending on the users input, the program will proceed to the 'if' statement which says: 'userInput' is equivalent to number 'x' (x is the corresponding number). Then it runs the code, which is in that statement.

Choice 1 – Making an appointment

Before the program enters the 'if' statement, it has prepared a struct, which serves as a basic structure for reference. It makes a struct named 'p' from the basic. Then it creates a vector, which the program uses to store the struct. Then the program enters the the 'if' statement and asks for some user input which fills out the empty values, in the struct. Then it puts the struct to the beginning of the vector.

After this is done, the program has stored the assignment for later use. Then it changes the value of an integer to '1' which causes the program to enter another if statement which contains code similar to the menu code. This means that the code that is run for the startup menu and the code it goes to afterwards are not the same, but look similar. Then the program waits for a new user input.

Choice 2 – Checking appointments

After the program has gotten the user input '2' it will enter the check appointments code. When it begins, it starts off by clearing the screen with the 'system("CLS");' function. Now with a clear screen, it prints some information on an integer named the reference number.

With a for loop, the program both checks and prints (if there is anything to print) all structs stored in the vector. It does so by starting from the very first position in the vector, taking that specific struct and then printing it. Once it has done that it adds a value of 1 to the integer 'n' (which starts from 0) until 'n' has the same value as the size of the vector. The reference number aka. 'RefNr' is the value 'n', which means that the reference number really just is a structs position in the vector.

When there is no more structs to be printed it exits the for loop and then pauses the program (system("PAUSE")). Afterwards we change the value of 'userInputTwo' to '3' which causes the program to return to the menu. This is why we pause the program, otherwise the program would return to the menu, before anyone having a chance to see the choice 2 prints.

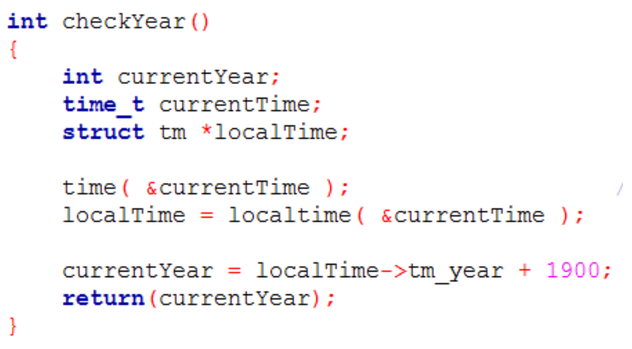
Choice 3 – Print current month

After the program gets the input '3' it enters the 'if' statement and immediately enters an infinite loop.

When it has entered it clears the screen with the "clear" function, then it sets up some integers it will need for later use. Then it sets the global integer 'year' value to that of the function 'checkYear()'.

In the 'checkYear' function it sets an integer for later use. Then it uses the 'time_t' to check how many seconds there has been since 1970 and it saves the value as 'currentTime'. Then it calls for the struct called 'tm' which is saved in the library 'time.h' which is included in the library 'ctime'. Then it uses the 'time' function, which gets the current calendar time of 'time_t' and adds that value to the 'currentTime', then it converts the 'time_t' library to the struct 'tm' as local time. Finally the function sets the integer 'currentYear' value to that of the 'localTime -> (specifically) tm_year + 1900' And finally returns the integer Current year, we are adding 1900 in order to obtain the correct year.

After it has checked the current year it prints it.

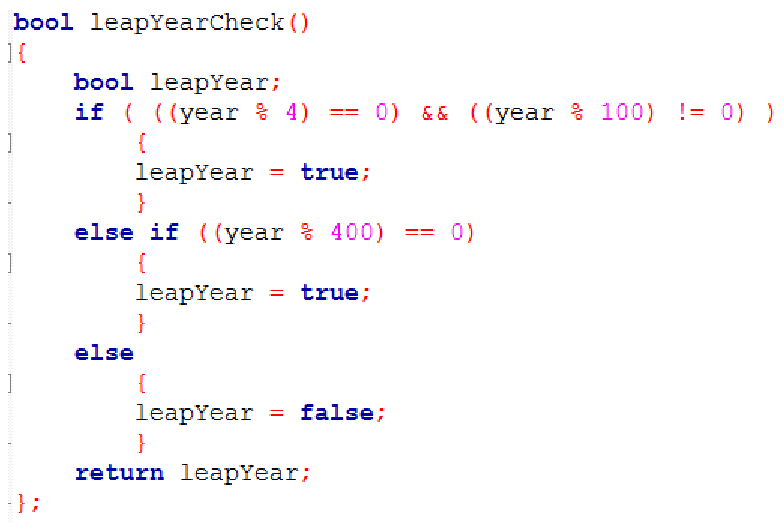


```
int checkYear()
{
    int currentYear;
    time_t currentTime;
    struct tm *localTime;

    time( &currentTime );
    localTime = localtime( &currentTime );

    currentYear = localTime->tm_year + 1900;
    return(currentYear);
}
```

Figure 2 The checkYear function

Then it start another while loop which will continue until we have done a process for every month in the current year. Now it checks the number of days there is in the current month, it does so by called the function named so and gives it the information of the current month we are checking (this value changes compared to which month it is checking).

In the 'numOfDaysInAMonth' function, it stores the data it was given as the integer 'm' and now it simply checks the 'if' statements in the function, then returns that value. If it hits the case of being February (the value of 2) it will have to check whether or not it is a leap year, so it calls the function called 'leapYearCheck()'



```
bool leapYearCheck()
{
    bool leapYear;
    if ( ((year % 4) == 0) && ((year % 100) != 0) )
        {
        leapYear = true;
        }
    else if ((year % 400) == 0)
        {
        leapYear = true;
        }
    else
        {
        leapYear = false;
        }
    return leapYear;
};
```

Figure 3 The leapYearCheck function

In this function, it takes the global integer 'year' and puts it against the parameters to see whether a year is a leap year. These parameters are: The year has to be a multiple of 4, if it is, it must not be a multiple of 100, unless that number is a multiple of 400. If the year follows these parameters the function will return 'true', if it does not it will return 'false'.

Once it has checked the value from the 'numOfDaysInAMonth' function, it now checks if the while loop is at the right month to print by comparing the value to the 'checkMonth()' function. which does the exact same thing as the 'checkYear' function, except it now takes the '-> tm_mon +1' in the struct, the '+1' is because ‘tm_mon’ returns the value ‘0’ for january, so in order to get e.g. january which is the first month, we have to add one. Then it checks if the 'currentMonth' integer and 'checkMonth()' values are equivalent, if they are, the program will start to print. First it will print a header and it does so by calling the function 'printHeader()' and giving it the number for the current month.

In the print header, it does the same as in the 'numOfDaysInAMonth' function, but instead it prints month it currently is and it prints a basic layout for weekdays for the month as well.

After having printed the header it will print the amount of days and print them in the correct location, it does so with the 'printMonth()' function and here the function is given the 'numDays' integer.

In the 'printMonth()' function it set up two additional integers, 'day = 1; and weekDay' . Then it sets 'weekDay' to value of the function 'dayNumber()' and this function gets three conditions it has to account for, first 'checkMonth()', then 1, because we want the first day on a month and finally 'checkYear()'.

The 'dayNumber()' functions follows Zeller's algorithm for checking days.

$$w = \left(d + \left\lfloor \frac{13(m+1)}{5} \right\rfloor + y + \left\lfloor \frac{y}{4} \right\rfloor + \left\lfloor \frac{c}{4} \right\rfloor - 2c\right) \bmod 7,$$

Figure 4 Zeller's algorithm [2]

Once the day is calculated, the program subtract 1 from the value, because Zeller's algorithm works from 1 to 7, while the machine work from 0 to 6.

Now it calls for the 'skipToDay()' function, which with help from the 'skip()' function sets the distance from the first open spot. This makes sure that the program places the correct day on the correct month and that the months start on the correct days. Finally the program will put in the days until the 'day' value is equal to the 'numDays' integer.

After this is done, it prints the current time and then then it waits for a response from the user. If there is a response, (random keyboard hit) it returns to the menu.

Choice 4 – Change an appointment

If a user typed something wrong or the appointment is not as it should be, it becomes necessary to be able to change that specific appointment. If the user enters '4' the program will start off by clearing the screen and printing out that it needs a reference number for the specific appointment the user wishes to change. After entering the reference number, it will

[2] https://en.wikipedia.org/wiki/Determination_of_the_day_of_the_week#Zeller.E2.80.99s_algorithm (30-12-2015)

print out that appointment, so that the user can check whether what he entered was correct.

The program now just waits for the user to hit any button on the keyboard before it can proceed. If a keyboard hit is detected, it clears the screen again and then does the exact same thing as in the choice one code, except it does not put the struct in the vector, because it is just changing the values of the struct that is already within the vector. After that, it returns to the menu.

Choice 5 – The alarm

If the user enters the number '5' the program will immediately enter an infinite loop (while(true)). This because when the code enters this program, it is supposed to just do this process again and again, until a user wants to add another appointment, or change one. When it has entered the while loop, in then calls for the void 'checkTime()'.

The very first thing the void does is checking how many seconds there has been since 1st of January 1970 at 00:00 o' clock, this is the due to the 'time_t' function. Then it creates a string which is immediately manipulated with the 'ctime' function, so that the string now will print in a certain way. Because of this, we now know the size of the string and we have the time stored in that string, some new variables are declared in order to work with the specific values for hours, minutes and seconds. Then it makes another array which copies the string array in order to get the clock in that array and then it prints it, has delay on approximately 1 second (real time) and ends the void.

Back in the code for choice 5 after having updated itself with the time it then proceeds to a for loop, which does so that the program will do the following action as many times as there are structs in the vector, in order to check all of them. Then it comes to an if statement, which now checks everything related with time in the current struct in the for loop. If all of those values match with the current time, it will then tell the movement software to start moving. If any of the variables does not match, it ignores the 'if' and then checks the next struct in the vector. This sequence will continue forever, unless a button on the keyboard is hit (if (kbhit)) it will then 'break' the infinite loop and return to the menu.

Movement

In the following the algorithm, will be explained on the basis on a flow chart and point by point text.

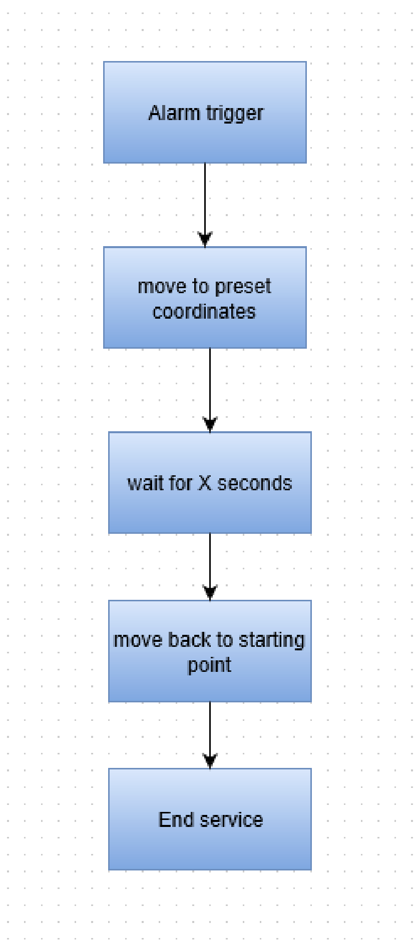


Figure 5 Flowchart for the movement part of the program

- Alarm trigger:

  The calendar node will have an alarm start at a given time, the calendar node will then communicate to the movement node to start.
- Move to preset coordinates:

  The movement node will have coordinates set in the program, based on the pre made map. based on this the robot will move, while avoiding obstacles.
- Wait for x seconds: When the robot have move to the coordinates, it will then wait for a set number of seconds.
- Move back to starting point:

  When the robot has reached its target position it is important that the robot goes back to its starting position, this should be done after a brief moment, but not quicker than it is possible to recognise that it has reached its goal.
- End service:

  The robot will then finish its program and be ready for the next trigger.

Code

The program will be initiated by the calendar and the movement program will then tell the robot to go to a specific point depending on where the task requires it to go, then do a task while there and then return to its starting point.

The first thing we did in the code was to create a section where we ask the program to listen for messages over the channel chatter, if it receives a message over the channel the action client will be told to spin a thread with the command ‘MoveBaseClient’ and then we allow the action server five seconds to come up, otherwise an error will occur. The next thing that happens is that a goal is created and that goal is sent to the 'move_base' with the ‘move_base_msgs::MoveBaseGoal’, this basically means that the base receives information of where it should drive. The frame_id is set to 'map' which allows the program to use coordinates from the map to send a destination. The next thing is to create a goal is the coordinates which is published with the 'goal.target_pose.pose.position' followed with either a .x or .y coordinate and these are the desired location for the goal the robot should move to.

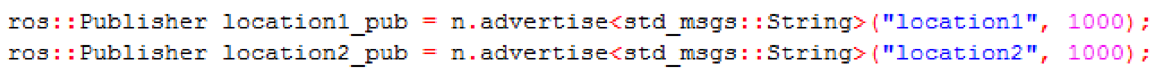
The last thing to do is to send the goal, which is done with the line for sending a goal called ‘ac.sendGoal(goal)’ and then it should start moving to where we want it to be, after the program ask for a confirmation when it gets to its target position. After all this it has to go back to its starting point and that is achieved by the program creating a goal the same way it did when it had to go to its target point. We have two movement nodes with each having a different goal but the same starting point, both nodes are connected to the scheduling program.

So in the end this program is able to make the turtlebot go to a specific point on a map that has been created, go to a certain spot and then return to its starting point.

Connecting nodes together

Allowing the different nodes to communicate, within the software, will be done with publisher/subscriber where our nodes will communicate over topics.

Topics is a method of messaging where nodes can subscribes to a topic, which means this node will receive messages published using that topic and other nodes can publish through this topic, meaning that they can send messages to the subscribed nodes. This system is defined as a “one to many” or a “many to many” communication model, meaning nodes cannot respond to messages received via topics.

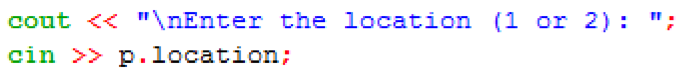


```
ros::Publisher location1_pub = n.advertise<std_msgs::String>("location1", 1000);
ros::Publisher location2_pub = n.advertise<std_msgs::String>("location2", 1000);
```

Figure 6 The two publisher chats

In the Scheduling program it is mentioned that it has set up two publisher chats, one named 'location1' and another named 'location2', this means that it is now able to send messages out to these two chat pubs and the nodes connected to this chatter.

In the movement nodes it is mentioned that they subscribe to one of the 'location' chats (each node subscribe to a their own chatter). The movement nodes run their code differently from the schedule program. The schedule program runs its main algorithm and depending on input, it switches to different sections of code with different functions. The movement program initialises



```
cout << "\nEnter the location (1 or 2): ";
cin >> p.location;
```

Figure 7 Where the user chooses location

their subscription to their chats and then with the 'ros::spin()' function, the program constantly checks if any of the 'chatterCallback' voids have received any messages. When they get a message from the publisher/schedule program, it initializes the movement algorithm in the 'chatterCallback' void. Afterwards, it continues the 'ros::spin()' loop until a new message is received.

Within the userInput 1, you can see the user chooses between one of two locations, this means if the user chooses the first location, it corresponds to the location1 publisherchat and therefore when the alarm trigger it will send a message to the node connected to it.
In userInput5 the node will check the time and if the alarm corresponds with the actual time it will then send the message depending on which publisher chat it corresponds to.