Specs and architecture V0.9

1. Physical

Dimensions intérieures chassis = 53 x 85

H axe roue avant = 9cm

H axe roue arrière = 12cm

Chassis OK sauf

* Coupleur moteur axe de 10mm sur la poulie d’entrainement, utiliser roue à visser plutôt que couplage axes
* Remplacer roues alus par roulements aciers/inox 12 x37x12, fixation sur arbre fixe (diam 6, l=20) avec 3 entretoises 4D15+12D12+4D15 - done

1. Architecture HW and key functions
   1. Overview

* Main unit (Pi) – I2C bus to TP (Traction and Position), ES (Environment Sensors) and BC (Battery Control)
  + Pi to manage connection with wifi dedicated AP for the front end
* Traction and position control board (Arduino Due 3v3) – I2C to sub boards
  + Compass (I2C 3v3)
  + GPS (tbd)
  + Decoders board (Arduino MEGA 2560 5V + level shifter I2C friendly)
  + Motor drivers (GPIO direct)
* Environment sensors board (Arduino 3v3, tbd) – direct or I2C to sensors
  + US sensor
  + Others (?)
* Battery control board (tbd)
  1. Main unit key functions
* MU1 - Webserver for HMI
* MU2 - Time reference
* MU3 - Maintain status database
* MU4 - Calculate trajectories
* MU5 - Drives Traction and position board (I2C-1 0x01)
* MU6 - Drives environment sensors (I2C-1 0x02)
* MU7 – Drives Battery control board (I2C-1 0x03)
  1. Traction and Position control board key functions

I2C-1 slave, I2C-2 master

* TP0 – manage emergency stop – (tbc, could go direct from MU to shorten execution loop, short circuit HW failures)
* TP1 – receive and decode order from MU

Order can be a segment (distance, speed) or a turn (target heading)

{“type”:”segment/turn”, “distance”:1.0, “speed”:100, / “target\_heading”:180}

* TP2 – calculate PWM x4 to execute order and set them
* TP3 – poll encoders (I2C-2) frequency = 0,1s
* TP4 – poll compass (I2C-2) = 0,1s
* TP5 – poll GPS (I2C-2) - tbc
* TP6 – adjust PWM to deliver order = 0,1s
* TP7 – feedback actual status (position (x,y), vx, vy, heading, speedL, speed R, power) to MU (I2C-1) = 1s
  + 1. Decoders
  + D1 – read (interrupt) encoder ticks

Interrupt overhead = 2,93micros before + 2,18micros after = 5,1125microsec in total + interrupt instructions time (ATmega 328 16Mhz) + 4,78microsec for a digital read or 1microsec for a register read = 7microsec – say x2 for overhead = 14microseconds. Will get 5\*70=350 interrupts per second per encoder => 4,900 microseconds = 5 miliseconds per second (0,5% per encoder). Sounds OK !

@ 350 interrupts

Cycle = 3ms

* + - Interrupt management: 14microsec x4 = 56microsec
    - I2C reading for 4 x2 bytes = 1ms (source: compass test) @ 10hz => 1ms/30 = 30microsec
* CPU load of 86microsec / 3ms = 2,8% = ok
  + - Leadtimewise, I2C to be executed between external interrupts. Should be fine because CPU is free 98% of the time  
      Mega Board (5V) able to manage 6+ external interrupts. Need to confirm the queeing of interrupt will not delay I2C call forever + latching is ok
  + D2 – count / decount based on direction

Interrupt on encoderA (raising, falling), check encoderB for level and increment/decrement

* + D3 – provides value since last poll

Time based interrupt (0,1s), latch data

Act as an I2C slave and deliver the data to PT

Issue = difficult to synchronize if two decoders boards - use a change interrupt to cascade Time based interrupt of on board to the other.

* + 1. Compass
  + C1 – provides heading

I2C call from master

* + 1. GPS

tbd

* + 1. Motors drivers
  + MD1 – sets direction

Basic execution of PT sketch

* + MD2 – execute PWM

Basic execution of PT sketch

* 1. Sensors board

tbd

1. Detailed specifications
   1. A. Main Unit
      1. MU1 - Webserver for HMI

* Main HMI page
  + Status block
    - state of ICE engine
    - Battery level
    - Position X,Y (P2 position on a map)
    - Heading
    - Speed
    - Traction power
    - Errors & warnings messages
  + File load block
    - Active routing / area info - info
    - Select active routing / area button
    - Routing / active area upload button
  + Actions block
    - Start / pause / resume
    - Emergency stop - button
    - Manual operations – button
* Manual operations page (V2)
  + - Go to base - button
    - Adjust base speed (50-100%)
    - Manual remote control (to store waypoints and manual test)
    - Store way point
    - Tbc - Stop ICE engine
* Waypoints / mown\_area upload page
  + - select file from local drive dialog box
    - upload, check structure, confirm, place in relevant directory and make active or raise exception
* Active waypoints / area selection
  + - list of available files in each category (2 boxes)
    - make active
    1. MU2 - Time reference – need to be confirmed
    2. MU3 - Maintain status database

Query TP status (1Hz, SPI bus) and load sql database.

x, y, heading, PWML, PWMR, calculated speedx, calculated speedy

Query BMS status (1Hz, SPI bus) load sql database:

Battery level, battery temperature

Query SB status (1Hz, SPI bus) load SQL database:

tbc

* + 1. MU 4 – Calculate trajectories

2 modes = waypoints and area to be mowed

* + - 1. Way points
  + Load waypoints

Waypoints file structure (tbc)

{{“type”:”waypoints”},{“name”:“toto”},{“status”:”valid/invalid/unknown”}}

{{x,y},

…

{x,y}}

* + Calculate segments
    - Seg0\_origin=starting point
    - Draw straight line to next way point
    - Current\_block = next crossed block
    - Check for forbidden blocks (block crossed + 3x3 around)
    - If ok Segi\_end = current\_block else Segi\_end = last\_block
    - Add waypoint @ 90° left + 3 blocks from last ok block
  + Play segments (MU5)
    - 1. Mowed area mode

Based on and random action within mowed area

* + load mown\_area

Mown\_area file structure

{{“type”:”area”},{“name”:“toto”},{“status”:”valid/invalid/unknown”}}

{{x,y},

…

{x,y}}

* + head towards mowed\_area
    - Find closest point of mowed\_area as waypoint 1
    - Calculate route (waypoint mode)
    - Enter into mowed\_area
  + Mowing trajectory algorithm
    - To be implemented – check for forbidden blocks and redesign mowed area to exclude them
    - Alternative 1
      * Generate random heading with heuristic based on maximum number of unmowed blocks
      * Calculate waypoint as intersection of heading and mowed\_area border
      * Go back 1 block
      * Turn by random heading generated based on heurisitic
    - Alternative 2
      * Generate inner border of mowed area moving all waypoints by one block towards the inner (need to manage narrow areas)
      * Mow along this line
      * Repeat same logic
    1. MU5 – Drive TP board

Send first segment then when segment completed, send next segment.

TP to report on position every second and end of segment status one step before completion (MISO line).

* + 1. MU6 - Drives environnement sensors (SPI 2) TBC
  1. TP functionalities
     1. TP1 – receives and decodes order from MU (SPI bus)

Order can be a segment (distance, speed), a turn (target heading)

{“type”:”segment/turn”, “distance”:1.0, “speed”:100, / “target\_heading”:180}

* + 1. TP2 – calculate PWM x4 to execute order and set them

If (order[“type”]=”segment”) Set all PWM=100

TO DO add a learning process because straingth line maybe different from equal to 100

If (order[“type”]=”turn”) Set right\_pwm=50 & right\_pwm=-50

* + 1. TP3 – poll encoders (I2C slave) frequency = 0,1s

Timer interrupt retrieve EC data (# of ticks since last latch for each wheel - 1 tick = 3,1EE-3 m). @full speed (5 rps and 70 ticks per round) expected reported ticks in [-35;35] => [-64;64] or [0;128] = will fit one byte on I2C.

* + 1. TP4 – poll compass (I2C slave) = 0,1s

Timer interrupt retrieve compass data = current heading

* + 1. TP5 – poll GPS - tbd
    2. TP6 – adjust PWM to deliver order
* Execute after TP 4 or TP 5 (Encoder ticks and compass available every 0,1s)
* Check if segment is completed (distance < 2,5 cm or heading <2°):
  + Reset counters
  + Check go ahead received from MU
  + Start next segment
* Check if segment is expected to be completed in next iteration (distance < PWM \* 5 cm or heading < 5°):
  + If yes : report segment completed
  + Set PWM to 50% of current \*4
* Straight line management (forward and backward)
  + If max (current gap; abs(sumof(current gap and previous gap))) between slowest left and slowest right >= 2 ticks (6% deviation):
    - slowdown fastest side (function of the gap tbd)
    - Reset current gap #fixed so do not carry forward
  + Else if sumof(current gap and previous gap)=0 :
    - Reset current gap #current and previous cancel out => do not carry forward
  + If gap between front and rear
    - Slowdown slipping wheel to slowest wheel of the corresponding/both side(s)
  + If no gap front rear and abs(sum(current gap – previous gap) <=1:
    - Increase speed by 10% on all
  + Former\_gap=current\_gap
* Rotation management
  + Pour un chassis de 60 x 90, circonférence cercle circonscrit = 3,4m => vitesse de rotation ~ 52°/seconde => roation à faire mi-vitesse soit 26°/s soit avec une fréquence de 5hz sur le compas une resolution de l’ordre de 5,2° qui est 2x la precision du compas.
  + Check if abs(current\_heading - target\_heading) < 10° then PWM=25%
    1. TP7 – feedback actual status (position (x,y), heading, teta\_point) to MU (I2C-1) every 1s
* Calculate x, y, vx, vy, teta\_point
  + x, y in milimeters [-1e6, +1e6] => need a long [-2e9; +2e9] (4 bytes) – int are 2 bytes [-32k to +32k]
  + vx, vy in mm/s => [-500,+500] => int (2 bytes)
  + teta\_point in 0,1 deg/s @max speed = 52°/s => [-520;+520] => int = ok
* Report on I2C-1 bus - position (x,y), heading, speedL, speed R
  1. Sensors board

To be confirmed

1. Data model
   1. Overall data model

Long = 4 bytes, int = 2 bytes

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Data | Type | Unit | Origin | MU | TP | ES | BC |
| x\_decoded | long | mm | Decode | x | x |  |  |
| y\_decoded | long | mm | Decode | x | x |  |  |
| vx\_decoded | int | mm/s | TP | x | x |  |  |
| vy\_decoded | int | mm/s | TP | x | x |  |  |
| bearing | byte | 360/255° | TP | x | x |  |  |
| Teta\_point | int | 0,1 °/s | TP | x | x |  |  |
| x\_GPS | long | mm | TP | x | x |  |  |
| y\_GPS | long | mm | TP | x | x |  |  |
| Vx\_GPS | int | mm/s | TP | x | x |  |  |
| Vy\_GPS | int | mm/s | TP | x | x |  |  |
| lat\_GPS | Tbd |  | TP | x | x |  |  |
| long\_GPS | Tbd |  | TP | x | x |  |  |
| Power | Int | mA | TP | x | x |  |  |
| ice\_status | byte | 0/1 | MU | x |  |  |  |
| Message | String (140) |  | TP | x | x |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |

* 1. Data model per board

1. SW architecture
   1. Main Unit software
      1. Front web app

* Based on Flask framework
* 3+1 Front end pages
* Classes
  + Rover
    - Data (x, y, speedx, speedy, messages, ice\_status, battery\_level, traction\_power)
    - Methods
      * query\_dbstatus
      * get\_status
      * save\_status
      * set\_routing – question to clarify how to pass it
      * emergency\_stop()
      * set\_speed()
      * Manual mode functions – not clear how to pass instructions real time to the motors in current architecture (manual\_forward(meters), manual\_backward(meters), manual\_turn\_right(degrees), manual\_turn\_left(degrees), set\_speed())
  + Routing
    - Data: type, list of waypoints
    - Methods: check, save, load\_from\_disk,
    1. Back end
    2. Data management

Class Routing()

* datas

mode, routing, next\_step

* functions

Class GPS()

Data

Functions

Initialization

launch webserver

mower=Mower()

while routing is not set :

pass

mower.create\_routing()

main loop

mower.get\_info()

check database for new orders

mower.calculate\_next\_step()

mower.execute()

2.4 Communication protocol

http

Robot connects to wifi

Robot Web server - Webpage of the robot