|  |
| --- |
| In this week we did a Odometry Calibration using a paper named **Calibration of omnidirectional wheeled mobile robots: Method and experiments.**  Calibrated scene shows : with x = 0.5 and c<133  x +1.0049e+00 (Ideally -1.00089954e+00)  y +6.8548e-03 (Ideally +0.11967e-03)  gamma +6.8066e-01 (Ideally 0)  **First I fine tuned given values for 1 meter right**  x=0.50199 and c < 132 , with this calibrated fine tuned scene shows x +1.0009e+00 (Ideally -1.00089954e+00)  y +6.1825e-03 (Ideally +0.11967e-03)  gamma +5.5369e-01 (Ideally 0)  So t= 132/20 = 6.6  Using This picture I measured angles:  Non calibrated robot:    This was my initial non-calibrated data measured 5 times:    My setAngularVelocityErrorsAndFlat function:  void setAngularVelocityErrorsAndFlat(){  angular\_velocity\_errors[0] = InverseJacobian[0][2] \* -1 \* error\_angle\_dot;  angular\_velocity\_errors[1] = InverseJacobian[1][2] \* -1 \* error\_angle\_dot;  angular\_velocity\_errors[2] = InverseJacobian[2][2] \* -1 \* error\_angle\_dot;  Flat[0][0] = 1 + angular\_velocity\_errors[0] / -3.99 ;  Flat[1][1] = 1 + angular\_velocity\_errors[1] / 2.0;  Flat[2][2] = 1 + angular\_velocity\_errors[2] / 2.0;    }  Data After Calibration:  radial error and the mean error improvement index :  corrected inverse Jacobian matrix:  Robot After Calibration:  To test my calibration I uncommented the inversejacobian part in setspeed and changed it with correctedinversejacobian:  void setSpeed(double x, double y, double theta) {  /\* double angular0 = InverseJacobian[0][0] \* x + InverseJacobian[0][1] \* y + InverseJacobian[0][2] \* theta;  double angular1 = InverseJacobian[1][0] \* x + InverseJacobian[1][1] \* y + InverseJacobian[1][2] \* theta;  double angular2 = InverseJacobian[2][0] \* x + InverseJacobian[2][1] \* y + InverseJacobian[2][2] \* theta;\*/  double angular0 = CorrectedInverseJacobian[0][0] \* x + CorrectedInverseJacobian[0][1] \* y + CorrectedInverseJacobian[0][2] \* theta;  double angular1 = CorrectedInverseJacobian[1][0] \* x + CorrectedInverseJacobian[1][1] \* y + CorrectedInverseJacobian[1][2] \* theta;  double angular2 = CorrectedInverseJacobian[2][0] \* x + CorrectedInverseJacobian[2][1] \* y + CorrectedInverseJacobian[2][2] \* theta;  ...    Video of part 1:  <https://youtu.be/pPRf7bQbhv0>  **First I fine tuned given values for double square motion**  double x = 0.50199;  double y = 0.0;  double theta = 0.0;  int c\_thresh\_right = 132;  int c\_thresh\_turn = 88;  move\_right\_1m(x,&robotino, &rate, c\_thresh\_right);  turn\_90\_degrees(&robotino, &rate, 1, c\_thresh\_turn);  move\_right\_1m(x,&robotino, &rate, c\_thresh\_right-6);  turn\_90\_degrees(&robotino, &rate, 1, c\_thresh\_turn+1);  move\_right\_1m(x,&robotino, &rate, c\_thresh\_right-6);  turn\_90\_degrees(&robotino, &rate, 1, c\_thresh\_turn);  move\_right\_1m(x,&robotino, &rate, c\_thresh\_right-6);  move\_right\_1m(x,&robotino, &rate, c\_thresh\_right-6);  turn\_90\_degrees(&robotino, &rate, -1, c\_thresh\_turn +1);  move\_right\_1m(x,&robotino, &rate, c\_thresh\_right-5);  turn\_90\_degrees(&robotino, &rate, -1, c\_thresh\_turn + 1);  move\_right\_1m(x,&robotino, &rate, c\_thresh\_right-6);  turn\_90\_degrees(&robotino, &rate, -1, c\_thresh\_turn + 2);  move\_right\_1m(x,&robotino, &rate, c\_thresh\_right-6);  Then I used corrected and uncorrected inverse jacobian and measured the results.  Graph for Before(BF) and After(AF) calibration:  radial error and the mean error improvement index for double square:  Video Before calibration :  <https://youtu.be/AARMeoHgR6w>  Video After calibration :  <https://youtu.be/hRuzLRTTFOE> |
|  | |  | | --- | |  | |