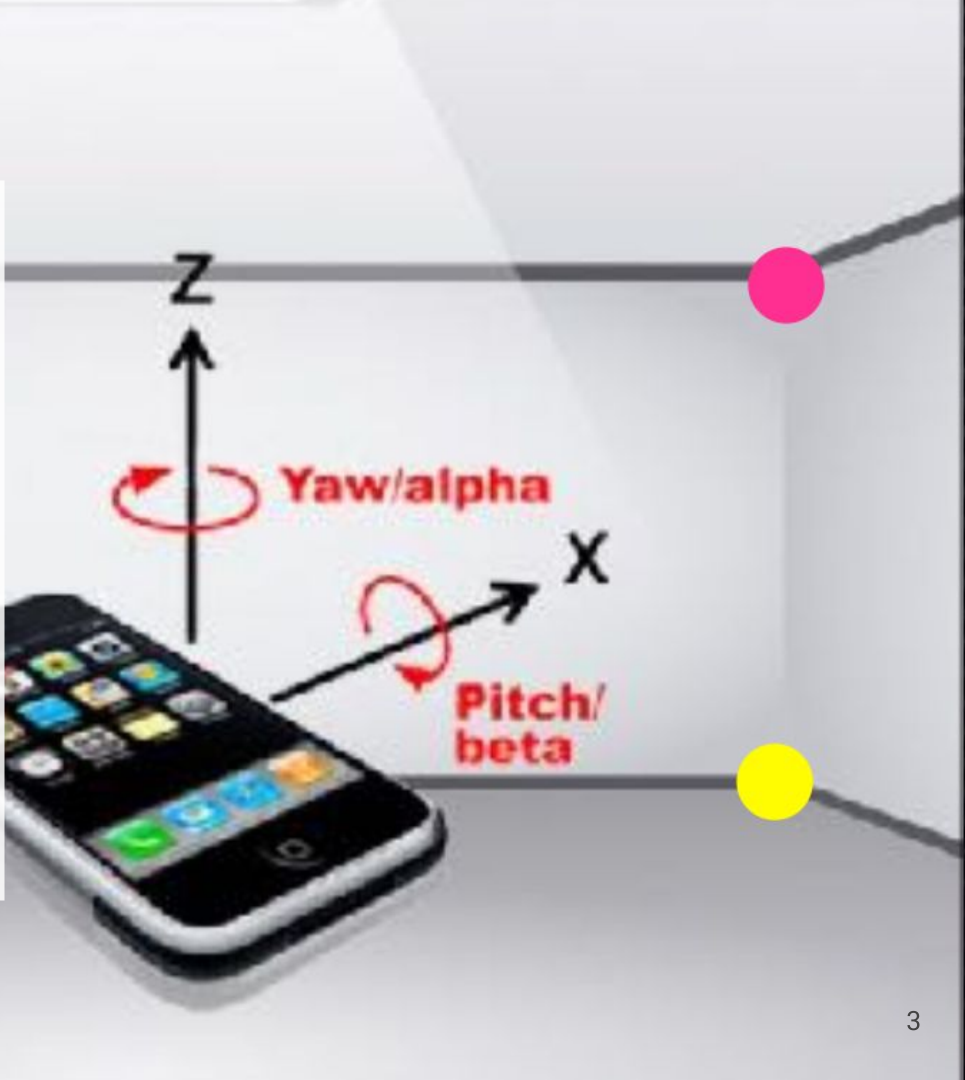
Visual odometry

Lawrence, Marc, Pol and Johanna

Contents

Problem Statement

Determining the **POSITION** and **ORIENTATION** of a **MOVING OBJECT** from the **IMAGES OF A CAMERA** attached to it.



Applications

- Robotics, computer vision
- Non-invasive position detection
- Sports (offside in football, dancing, ...)

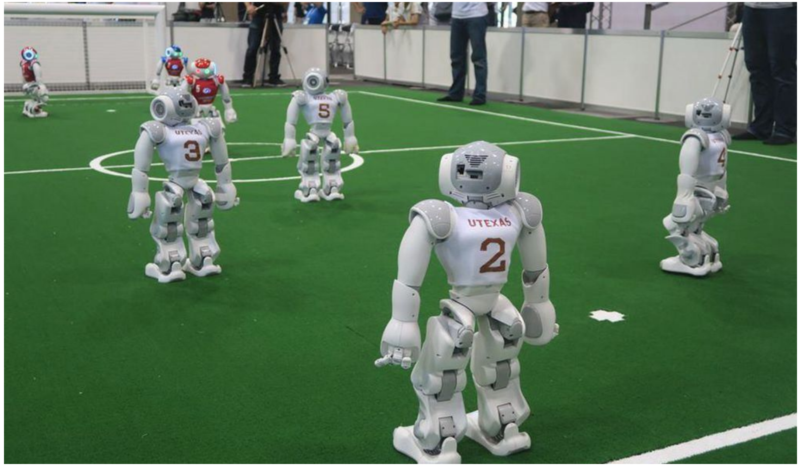
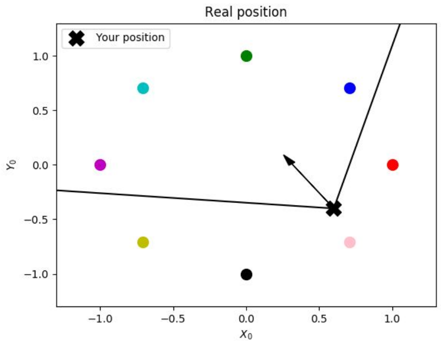


Image source:
https://www.geospatialworld.net/blogs/how-drones-are-changing-the-face-of-retail-sector/
https://www.smithsonianmag.com/innovation/why-funny-falling-soccer-playing-robots-matter-180964260/

Assumptions

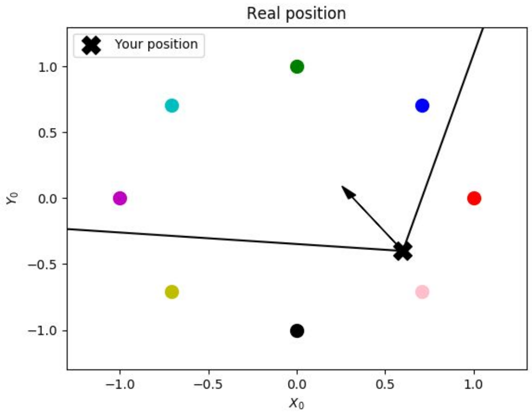
1. Environment of Object is predefined. Positions of objects in room are known.
2. Offset between object and Camera known, and constant
3. Height of camera predefined and constant → 2D
4. Object Position and Orientation parameters:
   a. Actual position (Composed of x, and y)
   b. Angle of view (theta)

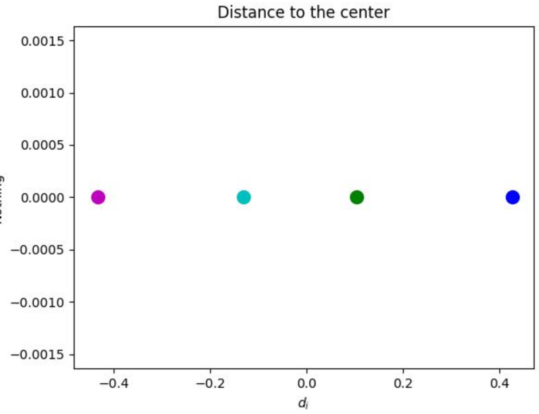


Methodology

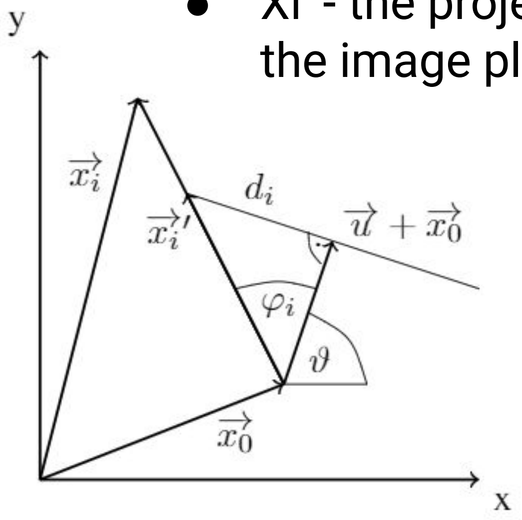
Analysis	Implementation 1	Implementation 2
Determine Relationship between image characteristics and object actual Position and Orientation. Direct problem	Use learned Relationships (functions) to estimate object actual position and orientation. Using geometry. Inverse problem - with Mathematical Modelling	Predict object actual position and orientation based on image characteristics using a trained machine learning model. Inverse problem - with ML

Problem in two dimensions





- Xo - our position
- U - the direction the photo is taken
- Xi - the position of the ball
- Xi’ - the projection of the ball onto the image plane



Direct Problem

From knowing the actual position to finding position on the image

We know :

- Our position
- The positions of the balls
- The direction of the photo

We want to find :

- The projection of the balls onto the photo

Inverse Problem

From the projection onto the photo find your position and the angle the photo was taken at.

We know :

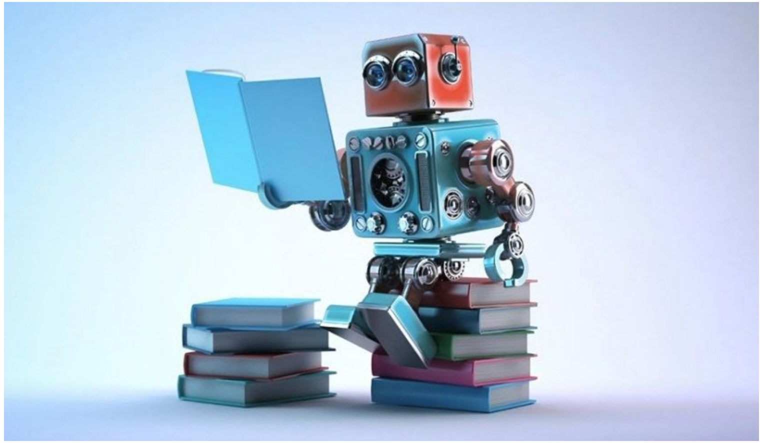
- The positions of the balls, their projections onto the image
- The focal length

We want to find :

- The position and the angle of photo

Approaches to Solving the Inverse Problem

1. System of equations
   a. Geometric approach
   b. Camera calibration
2. Machine learning



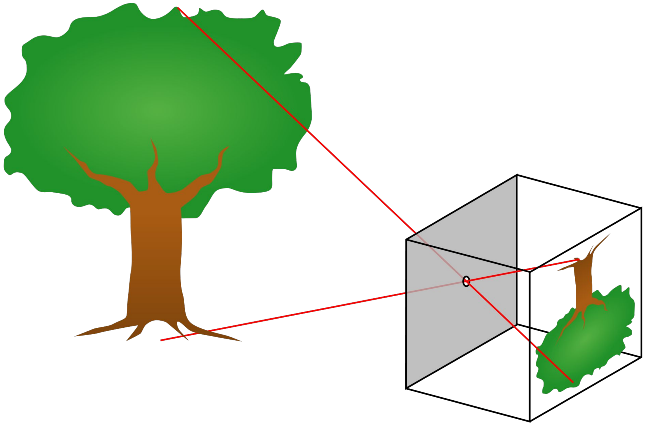
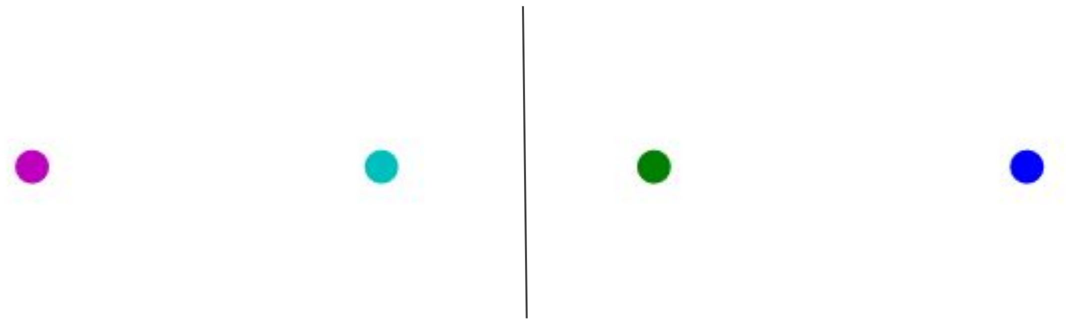


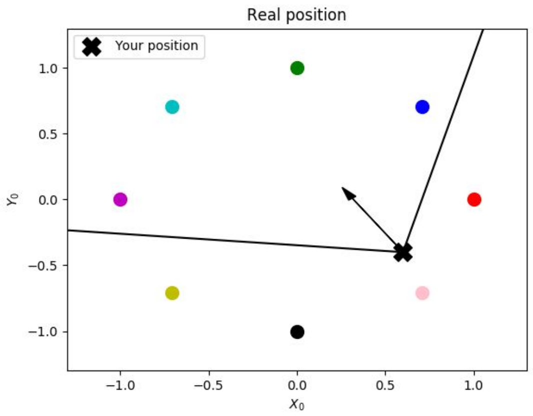
Image source: https://holyokecodes.org/events/machine-learning-without-coding/
https://en.wikipedia.org/wiki/Pinhole_camera_model#/media/File:Pinhole-camera.svg

Image extraction

Not only a theoretical project. We have implementation!



$$\{d_1, d_2, d_3, d_4, d_5, d_6, d_7, d_8\}$$



For this example

{nan, 0.4274, 0.1045, -0.1315, -0.4341, nan, nan, nan}

Geometric approach

$$\tan(\varphi_1 + \vartheta) + \frac{y_1 - y_0}{x_1 - x_0} = 0$$

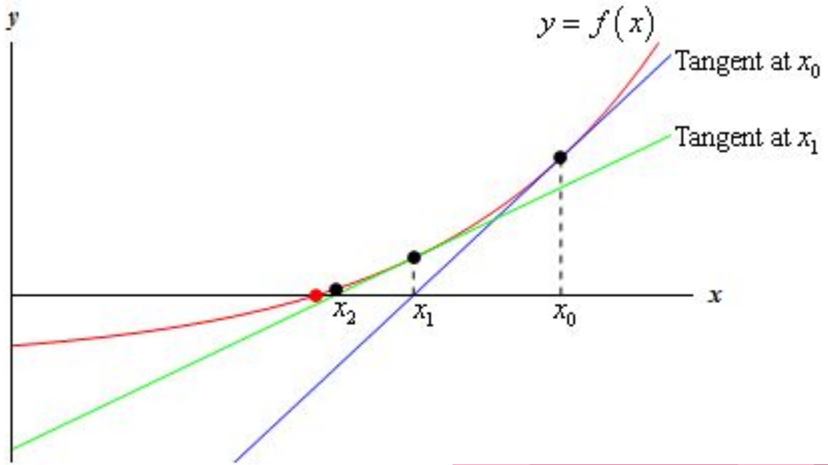
We have 3 unknowns, so we will need 3 points in the photograph in order to get 3 equations

$$\tan(\varphi_1 + \vartheta) + \frac{y_1 - y_0}{x_1 - x_0} = 0$$
$$\tan(\varphi_2 + \vartheta) + \frac{y_2 - y_0}{x_2 - x_0} = 0$$
$$\tan(\varphi_3 + \vartheta) + \frac{y_3 - y_0}{x_3 - x_0} = 0$$

Newton-Raphson method

Non linear equations, numerical solution using the Newton-Raphson method

$$x_{n+1} = x_n - J_F(x_n)^{-1}F(x_n)$$



The importance of the initial solutions!

Error on Convergence:

- In average if we move 0.001 for X_0 and Y_0, and 0.0005 for θ
- Radius 1 circle
- Probability of $2.5 \cdot 10^{-11}$ to choose a correct seed
- One success each $4 \cdot 10^{10}$ tries

Why?

- A lot of roots
- Gradient method to find the roots

Maybe Continuation Method works?

Continuation Method (Improvement of Newton)

Continuation Method tried for obtaining a good seed

We use a known system g=0 to solve f=0. $a \times f+(1-a)g=0$

Results:

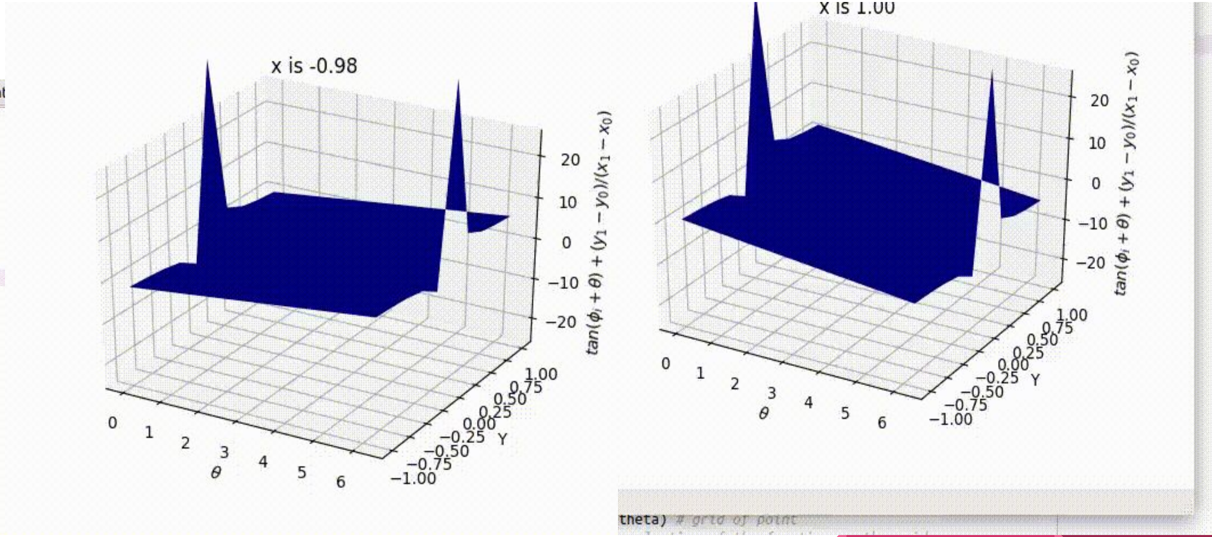
It doesn't converge to the correct result because it gets stop in some other root

Complexity of our equation

$$\tan(\varphi_1 + \vartheta) + \frac{y_1 - y_0}{x_1 - x_0} = 0$$

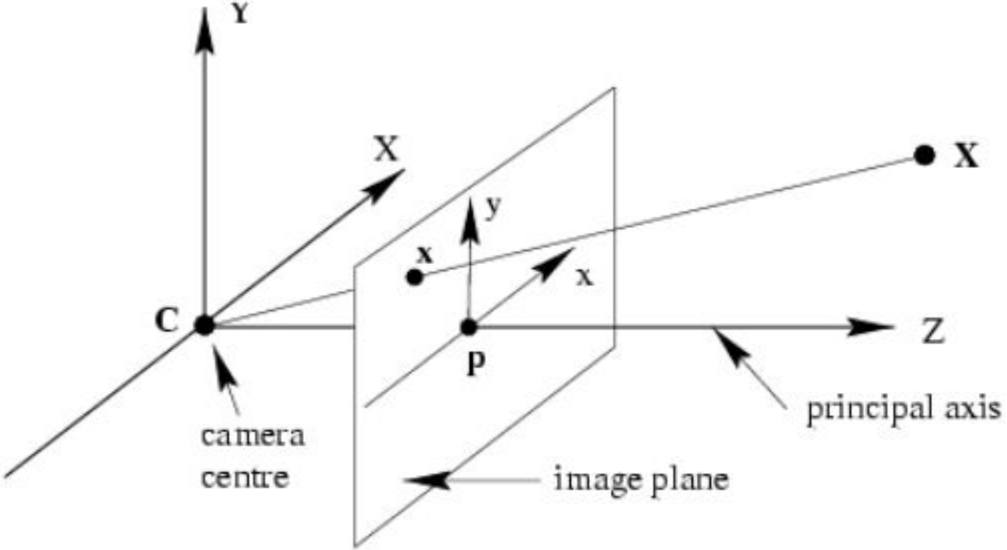
seline.py ✖ diffEq2Fsolve.py ✖ diffEqDoesASolutionWork.py ✖ Cont

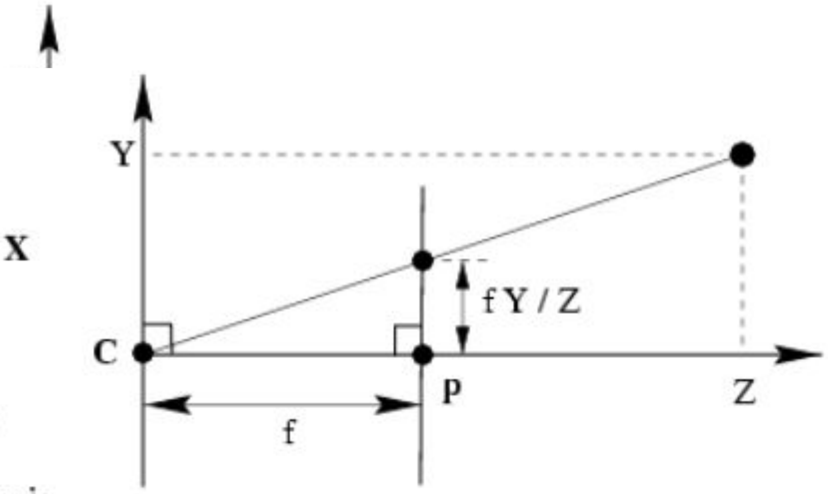
```
or i in range(len(x_range)):
    theta=np.linspace(0,2*np.pi,10)
    x = x_range[i]
    y = np.linspace(-1.0,1.0,10)
    X,Y = plt.meshgrid(theta,y) # grid of point
    Z = z_func_x(X, Y) # evaluation of the function on the grid

    ax.cla()
    #plt.pcolor(Z,vmin=-20,vmax=20)
    # Plot a basic wireframe
    ax.plot_surface(X, Y, Z, rstride=10, cstride=10, cmap='jet',)

    ax.set_xlabel(r'$\theta$')
    ax.set_ylabel('Y')
    ax.set_zlabel(r'$tan(\phi_i+\theta)+(y_1-y_0)/(x_1-x_0)$')


    ax.set_zlim([-25,25])
    #plt.colorbar()
    ax.set_title('x is {0:.2f}'.format(x))
    plt.pause(0.2)


_range=np.linspace(-1.0,1.0,101)
```



We need an easier equation!

Solving the Inverse Problem - Camera calibration approach





Camera calibration approach

$$\begin{bmatrix} cos(\vartheta) & -sin(\vartheta) & x_0 \\ sin(\vartheta) & cos(\vartheta) & y_0 \end{bmatrix} \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix}$$

$$= \begin{bmatrix} x_i cos(\vartheta) - y_i sin(\vartheta) + x_0 \\ x_i sin(\vartheta) + y_i cos(\vartheta) + y_0 \end{bmatrix}$$

$$x_i cos(\vartheta) - y_i sin(\vartheta) + x_0 - d_i x_i sin(\vartheta) - d_i y_i cos(\vartheta) - d_i y_0 = 0$$

$$\tan(\varphi_1 + \vartheta) + \frac{y_1 - y_0}{x_1 - x_0} = 0$$

$$\tan(\arctan(d_i/f) + \vartheta)$$

Clearly more linear!

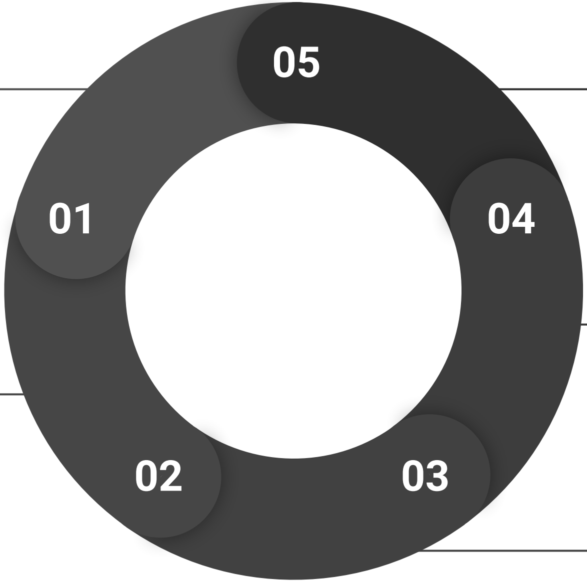
Analytical Solution

We are solving this system of equations

$$(x_1 - d_1 y_1)\cos(\vartheta) - (y_1 + d_1 x_1)\sin(\vartheta) + x_0 - d_1 y_0 = 0$$
$$(x_2 - d_2 y_2)\cos(\vartheta) - (y_2 + d_2 x_2)\sin(\vartheta) + x_0 - d_2 y_0 = 0$$
$$(x_3 - d_3 y_3)\cos(\vartheta) - (y_3 + d_3 x_3)\sin(\vartheta) + x_0 - d_3 y_0 = 0$$

By separating x0 and y0 from the last two equations and substituting into the first one we get $a\cos(\vartheta) - b\sin(\vartheta) = 0,$ where a and b are constants. From here we get theta and also x0 and y0 by using the known value of theta

Solving the Inverse Problem - Machine Learning Approach



Obtain Data

- Obtain training and testing data from processing several images

Data Pre-Processing and Analysis

- Exploratory Data Analysis

Preparation and Evaluation of Machine Learning Models

- Preparation of training, test, validation sets
- K-fold cross validation
- Evaluation of different ML algorithms

Finalize Model

- Choose best algorithm for final model
- Fine-tune algorithm parameters
- Prepare final model

Make Predictions

- Make Predictions on validation set
- Make predictions on test data

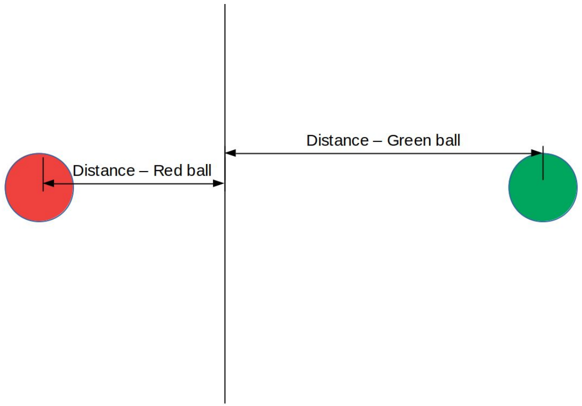
Obtaining Data

Images generated based on equations from solving the direct problem

Images were processed to retrieve required information and create dataset.

Information generated from images:

- Balls visible in image
- Distance from image centerline to center of balls



Camera view - centreline

red	blue	green	cyan	magenta	yellow	black	pink	x0	y0	Theta
NaN	NaN	0.374014	-0.416549	NaN	NaN	NaN	NaN	-0.398371	-0.054269	1.542349
NaN	NaN	0.374014	-0.416549	NaN	NaN	NaN	NaN	0.482887	-0.019398	2.232313
0.363488	-0.022626	-0.412821	NaN	NaN	NaN	NaN	NaN	-0.474068	-0.292890	0.714516
-0.054330	-0.416866	NaN	NaN	NaN	NaN	NaN	0.371825	-0.100153	-0.416075	0.301375
-0.400580	NaN	NaN	NaN	NaN	NaN	NaN	0.375536	0.281090	0.434339	5.892689

Known from Image

To be estimated

Preparation and Evaluation of Machine Learning Models

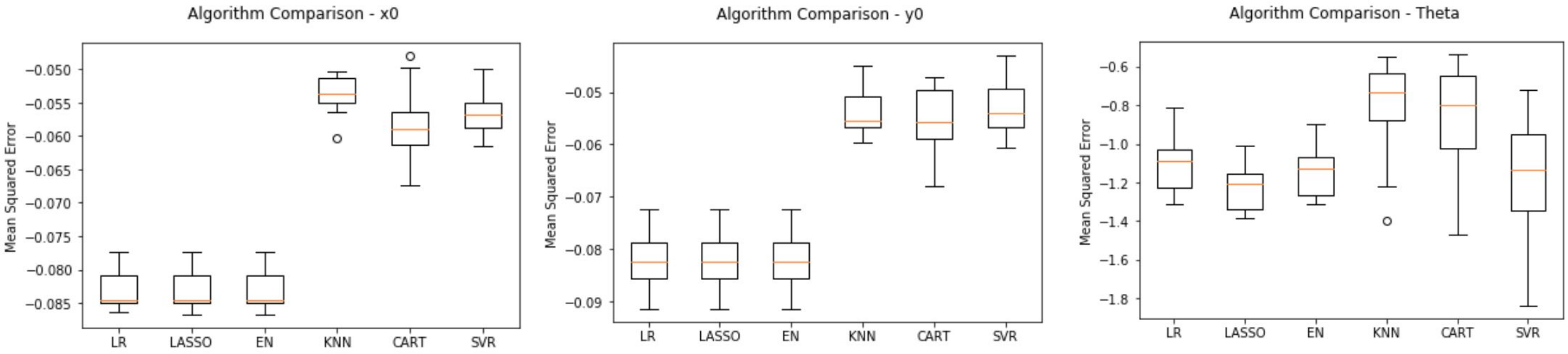
Model selection restricted to supervised and regression algorithms

Several machine learning algorithms considered:

- Linear Algorithms: Linear Regression, LASSO, Elastic Net
- Non-Linear Algorithms: KNN, SVM, CART

All six(6) algorithms trialed on the solution of the problem in order to pick the best performing model.

Evaluation of Algorithms

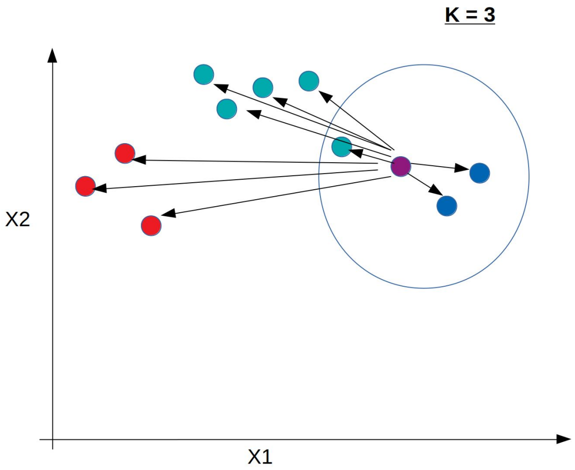


Separate models prepared to predict each variable

KNN chosen based on performance evaluated with MSE

K-Nearest Neighbor

For each test example, KNN algorithm outputs a continuous value which is the average of the values of its ‘k’ nearest neighbors



Parameters of KNN algorithm fine-tuned using grid search to identify optimum number of neighbors

```
Best: -0.049667 using {'n_neighbors': 21}
-0.085487 (0.009550) with: {'n_neighbors': 1}
-0.058771 (0.004125) with: {'n_neighbors': 3}
-0.053817 (0.002901) with: {'n_neighbors': 5}
-0.053492 (0.003172) with: {'n_neighbors': 7}
-0.052379 (0.003313) with: {'n_neighbors': 9}
-0.051125 (0.003344) with: {'n_neighbors': 11}
-0.050615 (0.003194) with: {'n_neighbors': 13}
-0.050151 (0.003185) with: {'n_neighbors': 15}
-0.050154 (0.003642) with: {'n_neighbors': 17}
-0.049916 (0.004185) with: {'n_neighbors': 19}
-0.049667 (0.004129) with: {'n_neighbors': 21}
```

Model Validation and Testing

Final Model has been validated with validation dataset and tested again on new unseen data.

Mean Square Error			
	x0	**y0**	**theta**
Validation	0.05 (2.5%)	0.05 (2.5%)	0.75 (12 %)
Testing	0.05 (2.5%)	0.05 (2.5%)	0.71 (12 %)

	x0_Actual	x0_Predicted	y0_Actual	y0_Predicted	theta_Actual	theta_Predicted	balls_visible
1	-0.035476	-0.084160	-0.178177	-0.196584	0.892034	0.884403	3
2	0.290907	0.333189	-0.312847	-0.310776	5.431745	5.368066	1
3	-0.089383	-0.103759	-0.096043	-0.075799	5.249794	5.180877	2
4	0.053209	0.048064	0.216779	0.236352	5.288363	5.263949	3
5	0.210150	0.232627	-0.048525	0.000213	3.137546	3.153378	3

Further work

- Time to do the computational implementation of the solution of the camera-calibration equations.

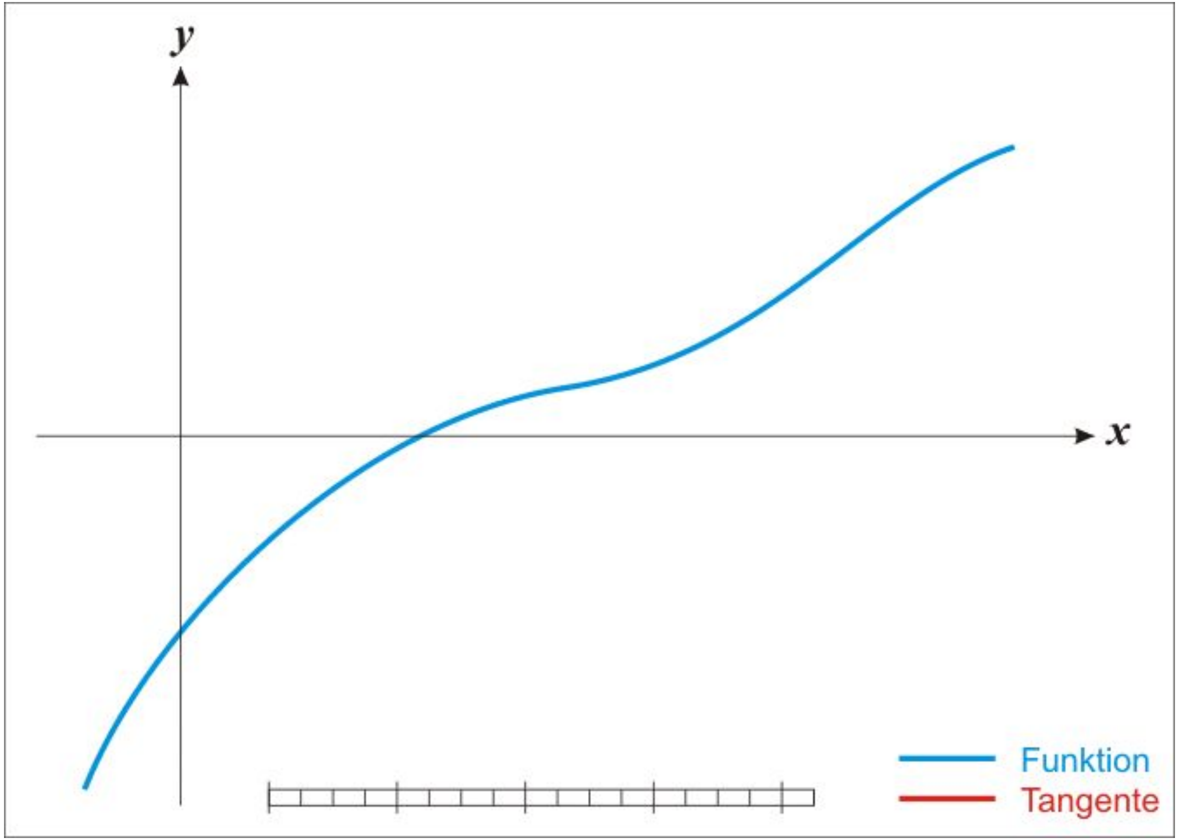
- Apply Machine Learning functions to predict position and orientation one depending on the other

- Barrier methods: to impose the solution inside the circle

Conclusions

- The direct problem works.
- We have tried 3 approaches for the inverse problem:
    - Trigonometrically: problems due to the non-linear function obtained
    - Camera calibration: better equation because it’s more lineal and the parameter f doesn’t appear
    - Machine Learning: works well (2.5% of x0 and y0 and 12% for theta). However it will be interesting to discover the real functions behind it.

END


y
x
Funktion
Tangente