

Motion Parameter eines 2-wheel Robot

mL.SetPwr(-100%...0...100%) .. Leistung \approx Geschwindigkeit

Vorw. mL.SetPwr(0,3); mR.SetPwr(0,3); V... Translatorische V-Komponente

Am Stand drehen: mL=+0,3; mR=-0,3 dPhi... Rotatorische V-Komponente

Kurve: mL=+0,3; mR=0,1; Trans + Rot

Simulation

rb.V... Translatorische

Trans: Rot

rb.dPhi... Rotatorische

Vorw.: rb.V=2,0; rb.dPhi=0; nur Translatorisch

Am Stand drehen rb.V=0; rb.dPhi=2,0; nur Rotatorisch

Kurve: rb.V=2,0; rb.dPhi=1,0; Trans+Rot

1b Kinematik eines 2-Wheel Robot

Bertl:

mL.SetPow(-100% . . 0% . . +100%) Leistung -> Geschwindigkeit

Vorwärts: mL.SetPow(0.3); mR.SetPow(0.3); **V** . . . Translations-Komponente des V-Vektors

Am Stand drehen: mL=0.3; mR=-0.3; **dPhi**. . . . Rotations-Komponente des V-Vektors

Kurve: mL=0.3; mR=0.1 Translation **V** und Rotation **dPhi**

Simulation:

rb.V Translations-Komponente

rb.dPhi. . . . Rotations-Komponente (Grad / FrameUpdate)

Vorwärts: rb.V=2.0; rb.dPhi=0.0; nur Translation

Am Stand drehen: rb.V=0.0; rb.dPhi=2.0; nur Rotation

Kurve: rb.V=3.0; rb.dPhi=1.0; Translation und Rotation

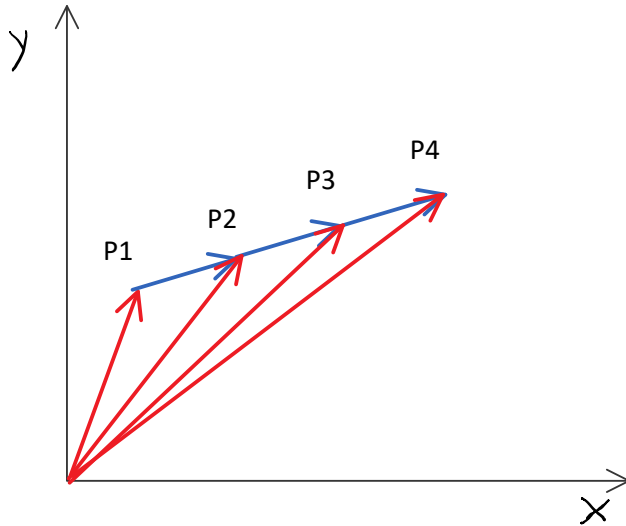
1b Ortsvektor und Geschwindigkeitsvektor fürs Geradeausfahren



Ortsvektor (wo ist das Auto)



Geschwindigkeitsvektor (wie schnell und in welche Richtung bewegt sich das Auto)



$$\vec{p}_{n+1} = \vec{p}_n + \vec{v} * \Delta t$$

Die Position zum Simulationszeitpunkt (n+1) ist
die Position zum Simulationszeitpunkt (n) + der Geschwindigkeitsvektor

ToTo:

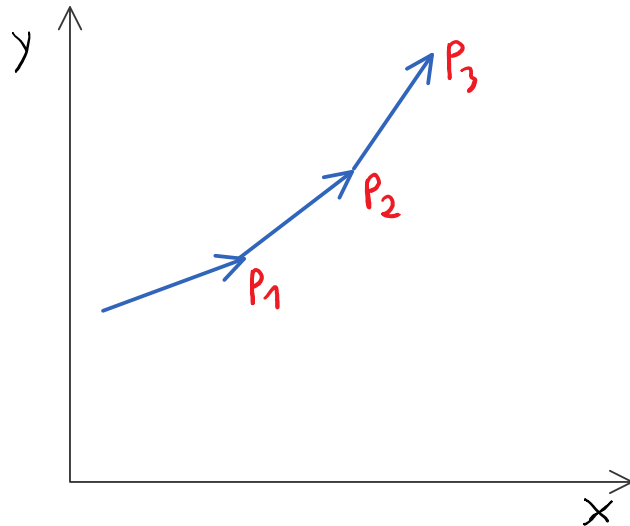
Diese Skizze für den Fall, daß der Roboter eine Kurve fährt.

D.h. der Geschwindigkeitsvektor rotiert mit der Winkelgeschwindigkeit **w**

2 Ortsvektor und Geschwindigkeitsvektor Kurvenfahrt

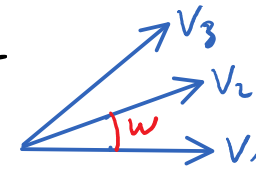
Verglichen mit dem vorhergehenden Fall dreht sich nun der Geschwindigkeitsvektor mit der Winkelgeschwindigkeit ω

- Ortsvektor (wo ist das Auto)
- Geschwindigkeitsvektor (wie schnell und in welche Richtung bewegt sich das Auto)



$$\vec{V}_{n+1} = \vec{V}_n \text{ rot } \omega * dt$$

$$\vec{P}_{n+1} = \vec{P}_n + \vec{V} * dt$$



Die Geschwindigkeit zum Simulationszeitpunkt **(n+1)** ist

Die Geschwindigkeit zum Simulationszeitpunkt **(n)** gedreht um ω

Die Position zum Simulationszeitpunkt **(n+1)** ist

die Position zum Simulationszeitpunkt **(n)** + der Geschwindigkeitsvektor

3 Die ITER PER TICK und dt Geschichte

$$x_{n+1} = x_n + v \cdot \Delta t$$

In allen unseren Bewegungssimulationen gibt es 2 wichtige Zeitgrößen

TIMER_INTERVAL das sind die Frames per Second

ITER_PER_TICK So oft werden die Bewegungsgleichungen pro neuem Frame durchgerechnet
Je größer ITER_PER_TICK gewählt wird desto genauer wird die Objektbewegung
(Physik) berechnet.

```
void OnTimer()  
{  
    for(i=0; i<ITER_PER_TICK; i++)  
        CalcNextPositions();  
}
```

Der Geschwindigkeitsmaßstab in unserer simulierten Welt ist *Pixel/FrameUpdate* und wird so gewählt, daß sich die Objekte mit einer beobachtbaren Geschwindigkeit über den Bildschirm bewegen.

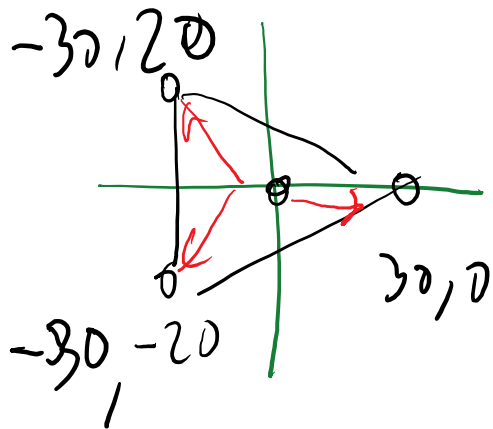
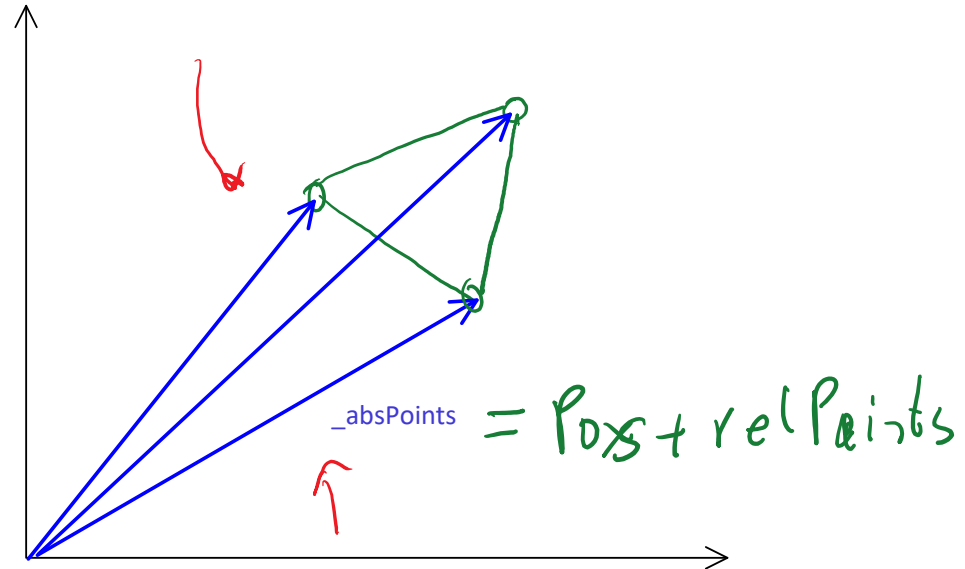
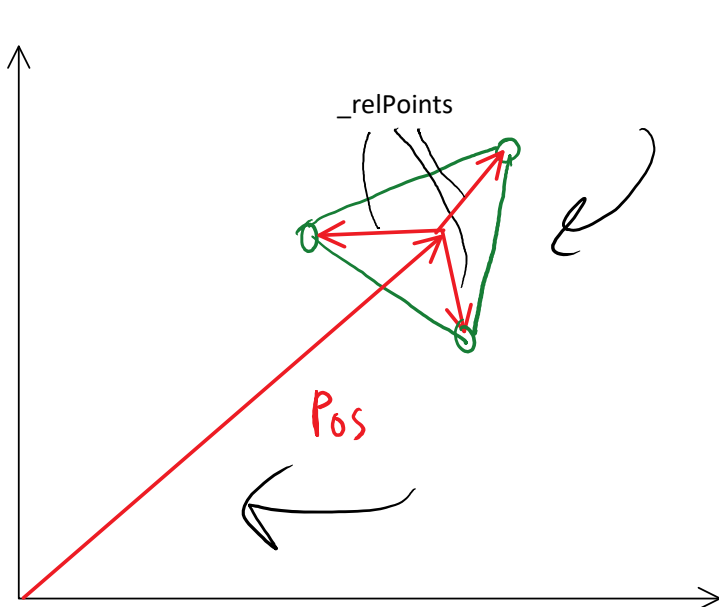
Wenn ITER_PER_TICK=1 ist spielt das **dt** für uns eigentlich keine Rolle.
Wenn wir allerdings ITER_PER_TICK verändern wollen (Rechengenauigkeit)
so müssen wir **dt = 1/ITER_PER_TICK** setzen damit die Objekte nicht unbeabsichtigt schneller werden.

Wir werden später noch komplexere Bewegungs-Differentialgleichungen kennen lernen
(z.B. Ball mit Reibung und Gravitation)

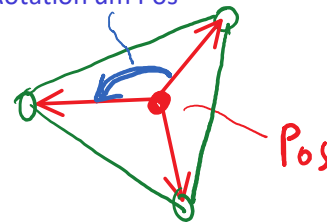
In all diesen Differentialgleichungen kommt das **dt** immer wieder vor und ist entsprechend richtig zu setzen.

4 Objekte rotieren

_relPoints beschreibt ein Vektorgrafik-Objekt relativ zur momentanen Position **Pos** des Objekts
durch die Rotation von **_relPoints** um den relativen Koordinatenursprung **Pos** kann die Richtung des
RotGraphicObj geändert werden

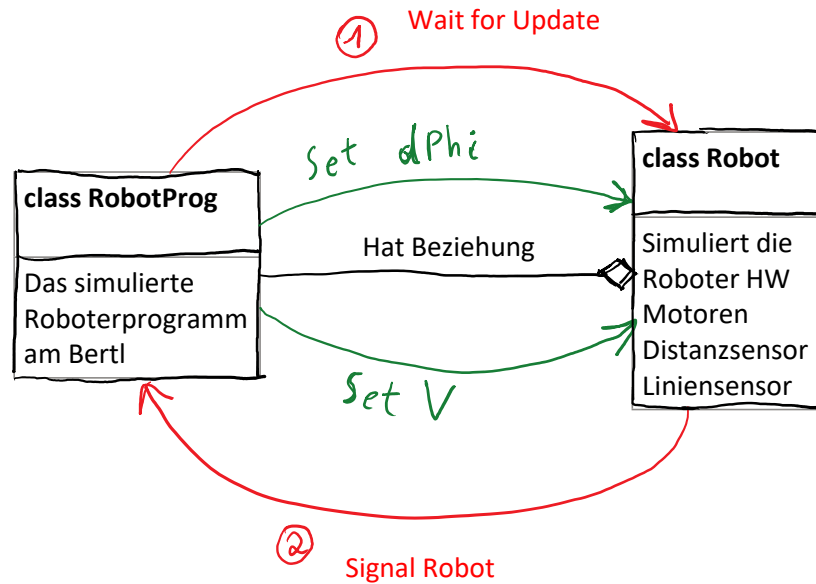


Rotation um Pos



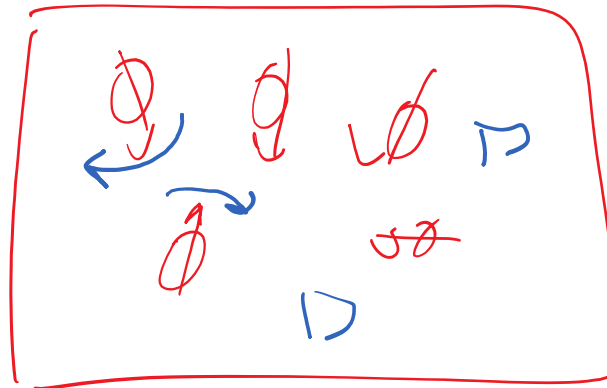
Code in [Bulgati.hl/SwDev4te/PhysSim/RotObj](https://github.com/Bulgati/hl/SwDev4te/PhysSim/RotObj)

5 Robot und RobotProg

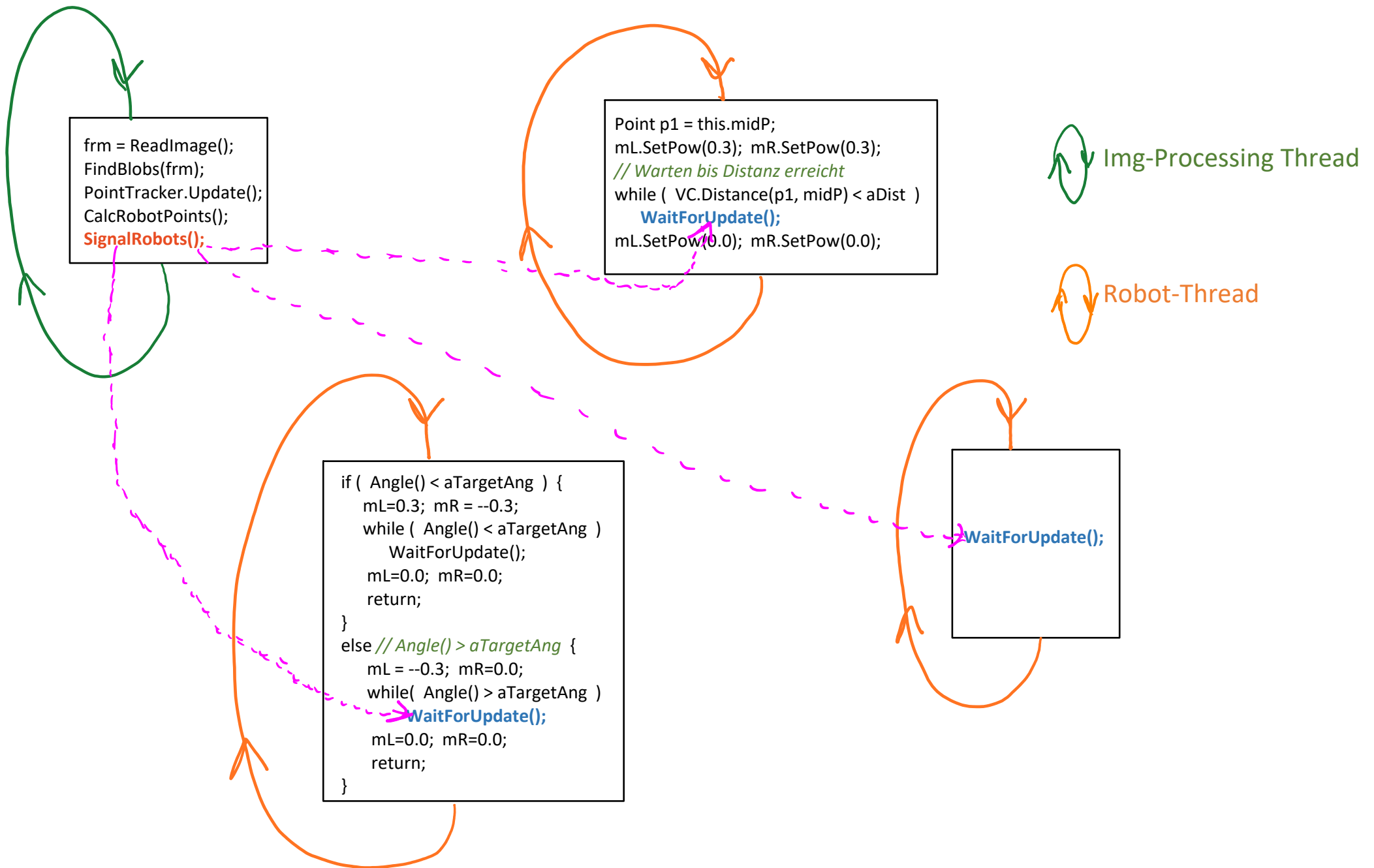


Das RobotProg blockiert am **WaitForUpdate()**-Aufruf und wird von der Simulation **aufgeweckt** wenn der Roboter um den nächsten Simulationsschritt weiterbewegt wurde.

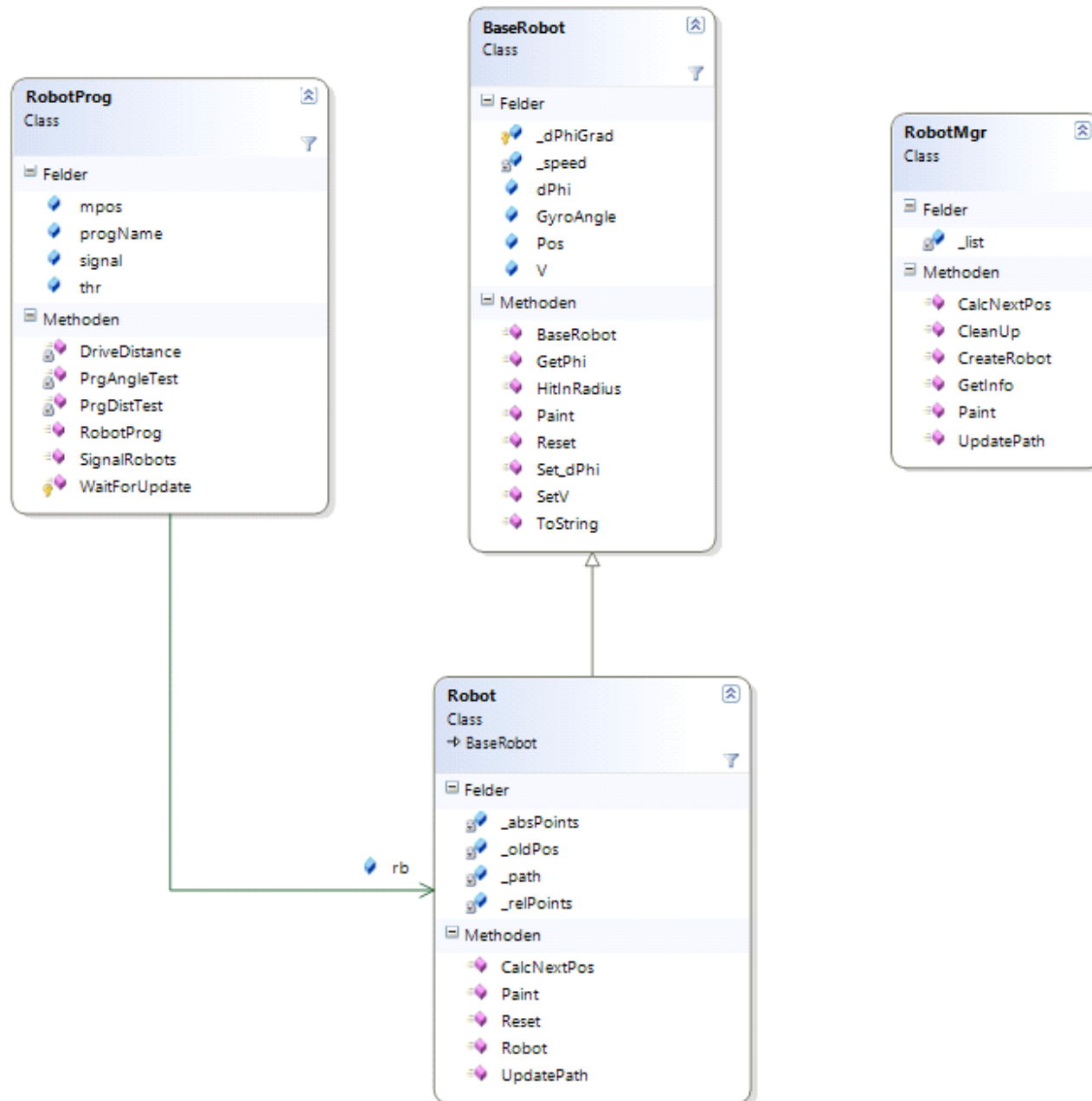
Das RobotProg setzt beim Roboter die Lineargeschwindigkeit **SetV()** und die Rotationsgeschwindigkeit **Set_DPhi()**. Das entspricht den Möglichkeiten die man am echten Bertl mit den Leistungseinstellungen für die beiden Motoren hat.



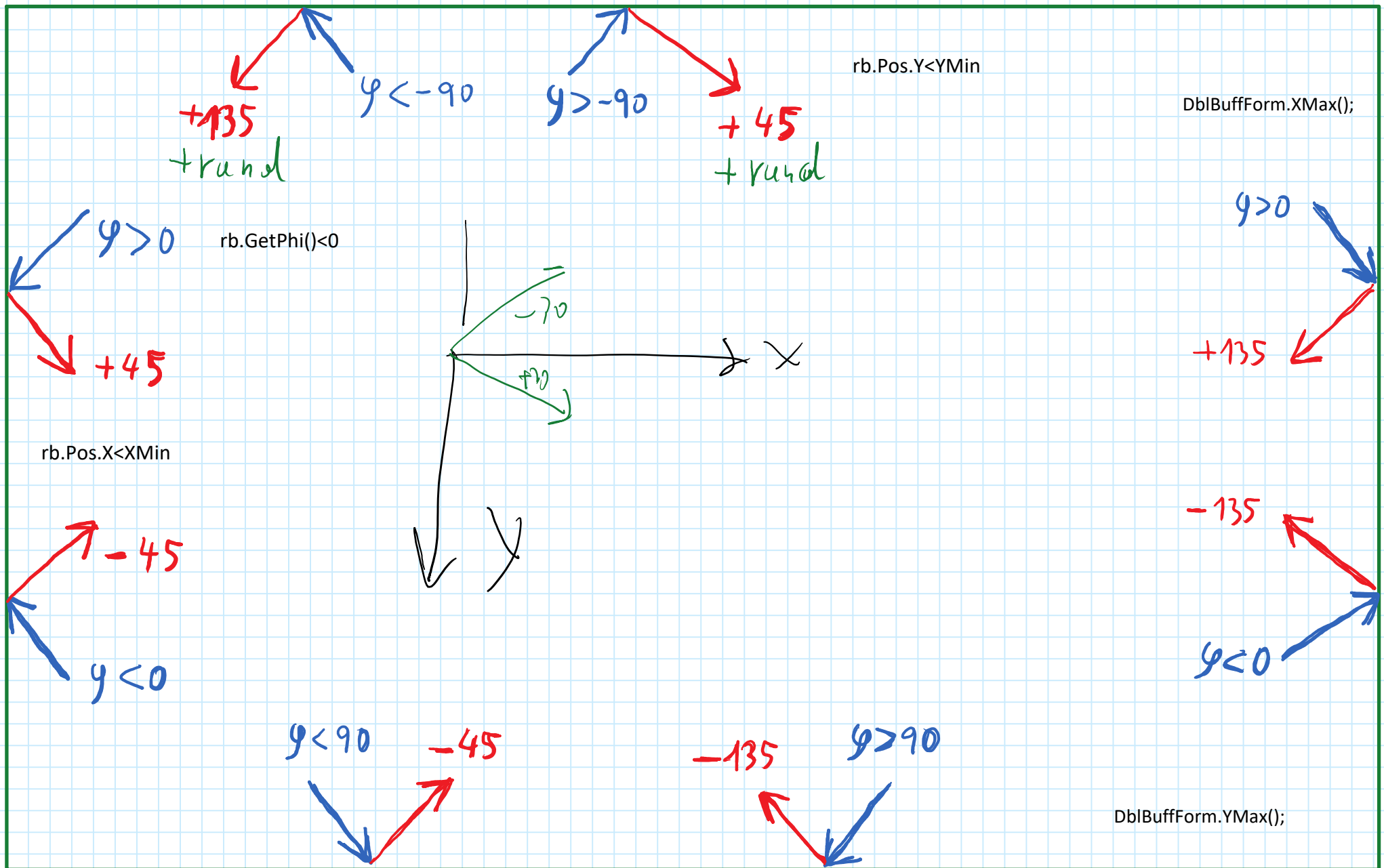
5a SW-Architektur Bildverarbeitungs PC



5c Robot und RobotProg

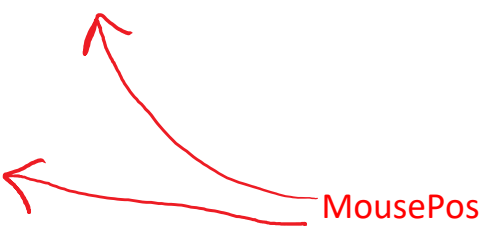


6 Reflect at Border



6a Reflect at Border some Tools and Tips

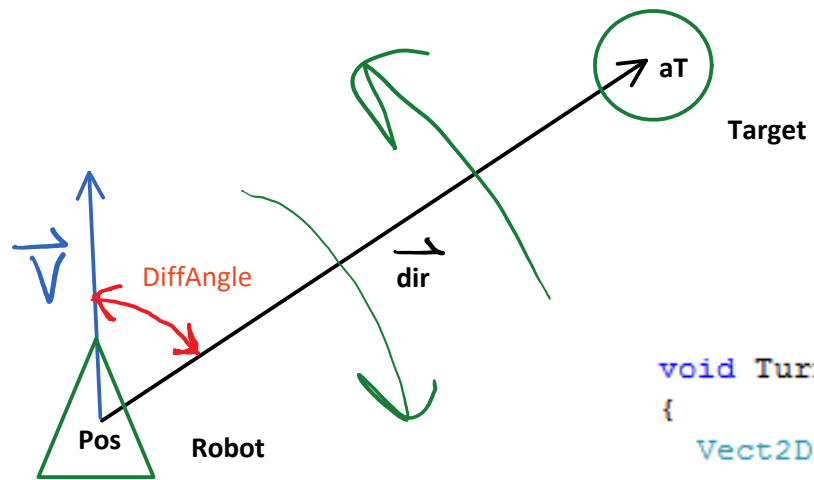
```
void PrgReflectBorder()  
{ // Rasenmäher  
    const double F_POW = 8; // Foreward  
    const double T_POW = 5; // Turn  
    const double R_POW = -2; // Reverse  
  
    // rb.Pos.X <= 0;  
    // rb.Pos.Y <= 0;  
  
    DblBuffForm.XMax();  
    DblBuffForm.YMax();  
  
    Vect2D tmp = mpos;  
}  
  
// oder  
RobotProg.mpos
```



MousePos



7 Turn2Target und DiffAngle



```
void Turn2Target(Vect2D aT, double aRotSpeed)
{
    Vect2D dir = Vect2D.Create(rb.Pos, aT);
    if (dir.DiffAngle(rb.V) < 0)
    {
        rb.Set_dPhi(-aRotSpeed);
        while (dir.DiffAngle(rb.V) < 0)
        {
            WaitForUpdate();
        }
        rb.Set_dPhi(0);
    }
    else // dir.DiffAngle(rb.V) > 0
    {
        rb.Set_dPhi(aRotSpeed);
        while (dir.DiffAngle(rb.V) > 0)
        {
            WaitForUpdate();
        }
        rb.Set_dPhi(0);
    }
}
```

8 ChangePlace

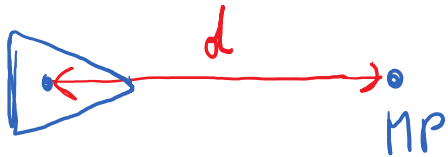
```
void PrgChangePlace()
{
    // wait until sim is started
    WaitForUpdate(); WaitForUpdate();

    int i1 = Omgr.GetNearestIdx(rb.Pos);
    int i2;
    if (i1 == 0) i2 = 1; else i2 = 0;

    while (true)
    {
        Turn2Target(Omgr.At(i1).pos, 5);
        Thread.Sleep(500);
        // TurnRelAngle(40, 5);
        Drive2Target(Omgr.At(i1).pos, 6, 4);
        Thread.Sleep(500);

        Turn2Target(Omgr.At(i2).pos, 5);
        Thread.Sleep(500);
        // TurnRelAngle(40, 5);
        Drive2Target(Omgr.At(i2).pos, 6, 4);
        Thread.Sleep(500);
    }
}
```

9a Maus nachfahren mit Regelungstechnik

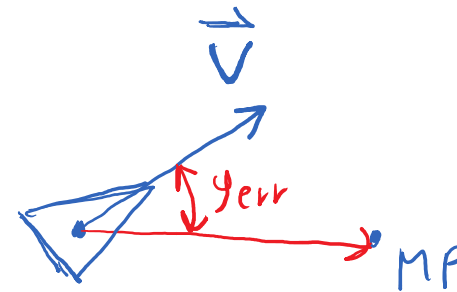


```
d = MP.DistanceBetweenPoints(rb.Pos);  
KP = 0.1;  
rb.SetV( d*KP );
```

Die Motorleistung (Geschw.) des Roboters wird abhängig vom Abstand zur Mausposition gesetzt. Je größer die Distanz zur Mausposition desto schneller fährt der Roboter auf das Ziel zu.

KP ist ein Verstärkungsfaktor (Tuningfaktor) der bestimmt wie stark der Abstand **d** auf die Motorleistung wirkt.

KP kann nicht beliebig groß gemacht werden da die Motorleistung begrenzt ist.



phiErr ist der Winkel zw. der Fahrrichtung des Roboters und dem Richtungsvektor zur Mausposition.

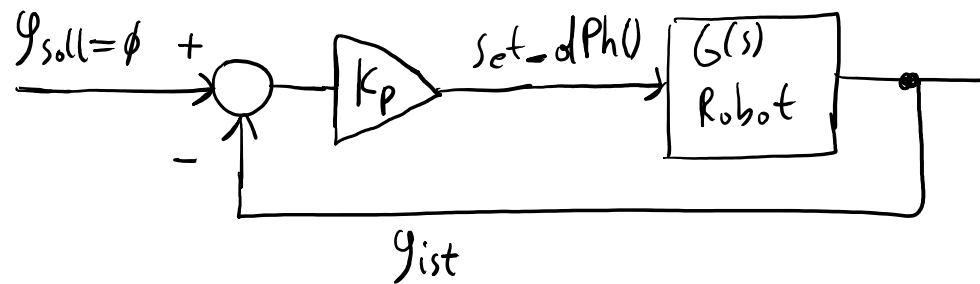
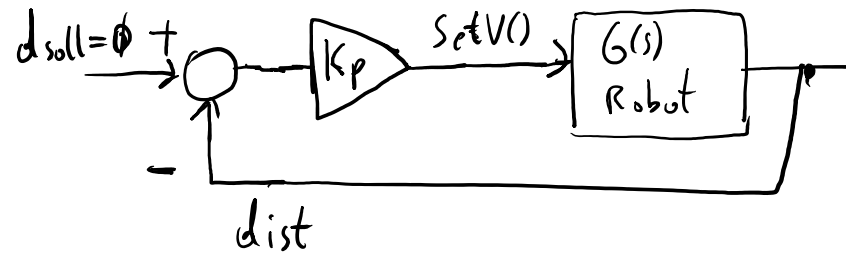
Mit **phiErr** wird über einen P-Regler die Winkelgeschwindigkeit **dPhi** des Roboters gesetzt.

```
KP = 0.2;  
dir = MP - rb.Pos;  
phiErr = dir.DiffAngle(rb.V);  
rb.Set_dPhi(phiErr * KP);
```

9 Teilaufgaben

- Rasenmäher simpel
- Maus nachfahren (Regelungstechnik)
- Maus Zug
- Ziele der Reihe nach abfahren
- 2 Roboter mit Antikollision
- Rasenmäher mit Antikollision (darf den Test verhauen)

10 FollowMouse Regelungstechnisch

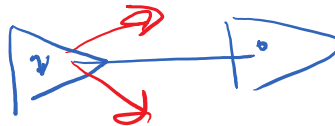


12 Robot Train

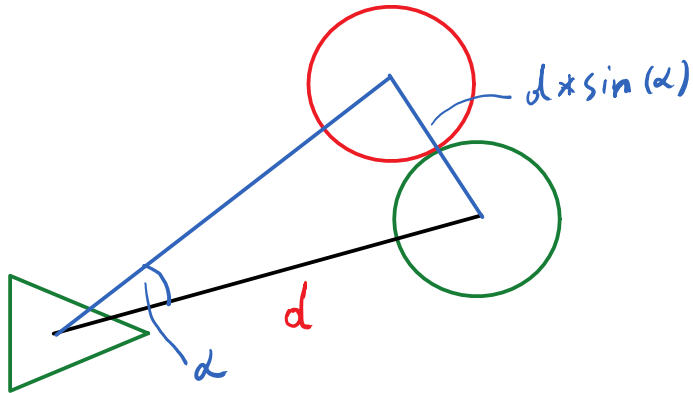


```
void PrgTrain()
{
    // warten bis die Simulation gestartet wurde
    WaitForUpdate();
    // Methode im RbMgr um den Vordermann zu finden
    rbFront = RbMgr.FindNearestInFront(this);

    while(true)
    {
        WaitForUpdate();
        // Die gleiche Regelungstechnik wie bei FollowMouse
        CalcSpeed(rbFront);
    }
}
```



13 Ausweichen Winkel und Abstandsberechnung

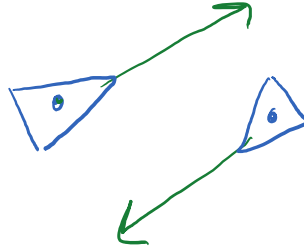


Wenn die Roboter den kritischen Abstand d erreicht haben.
Stehenbleiben und
Solange drehen bis
 $d \cdot \sin(\alpha)$ so groß ist, das die Roboter aneinander vorbeikommen.

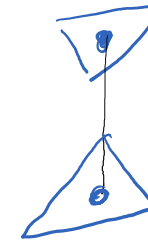
14 Ausweichen in 4 Schritten



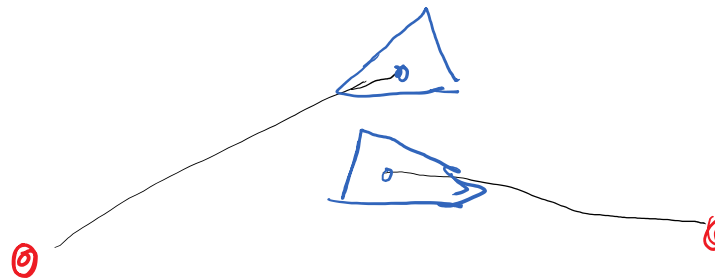
1. Wenn CollDist erreicht (unterschritten) ist sehen bleiben.



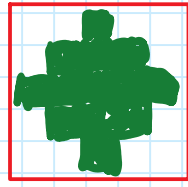
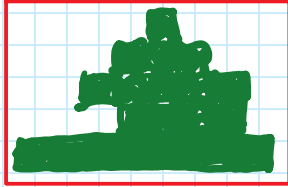
2. Drehen bis die Bedingung für das aneinander Vorbeikommen erfüllt ist

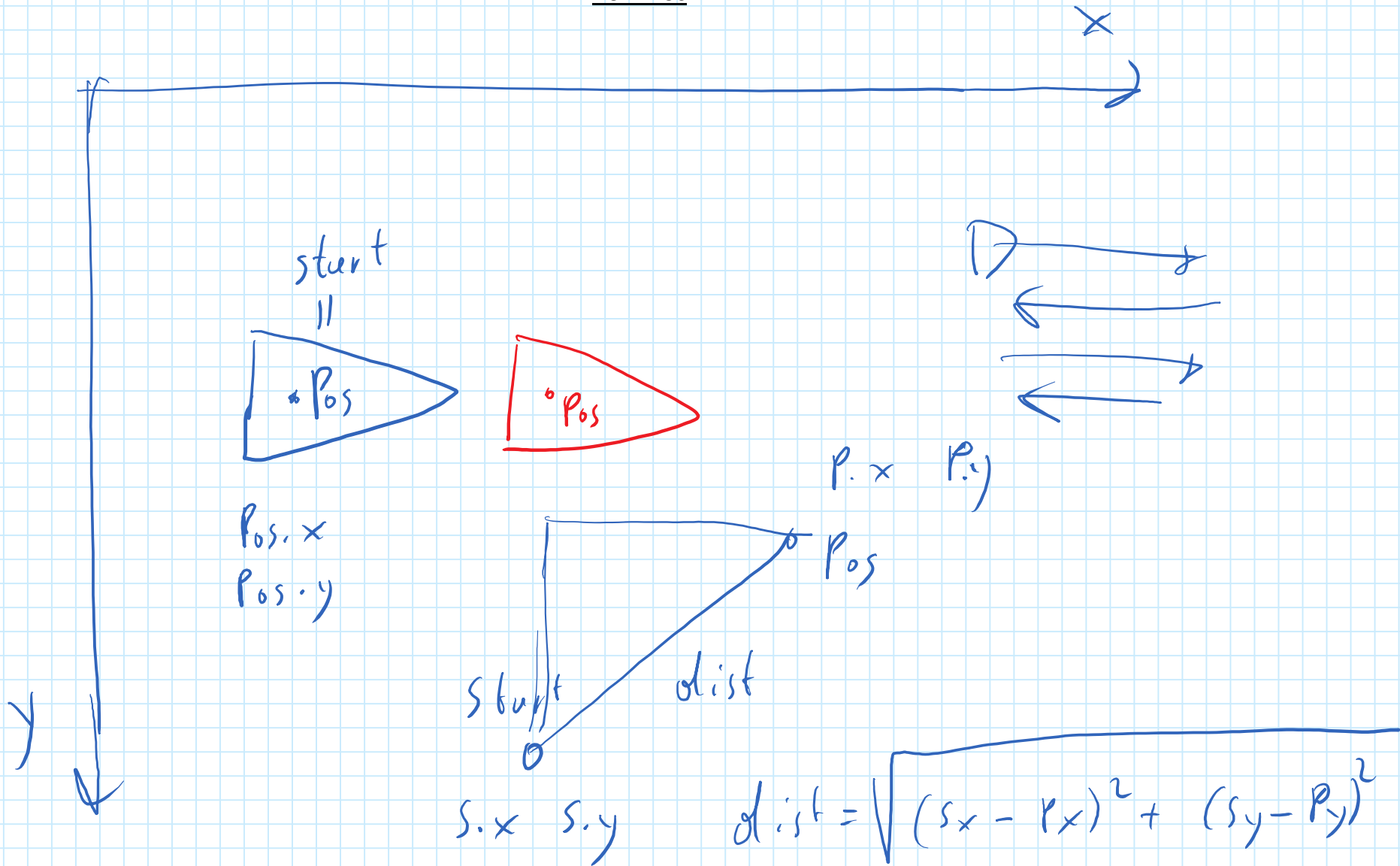


3. Aneinander vorbeifahren bis der oben gezeigte Punkt erreicht ist über X-Koordinaten feststellen



4. Wieder auf das Target ausrichten und weiterfahren





18 Kinematik

Kinematik

Bewegen, beschleunigen, bremsen, Kurve, drehen eines Bertl-Roboters in einer Simulation
2te annahme kann unendlich schnell beschleunigen

2D-Vektorrechnung

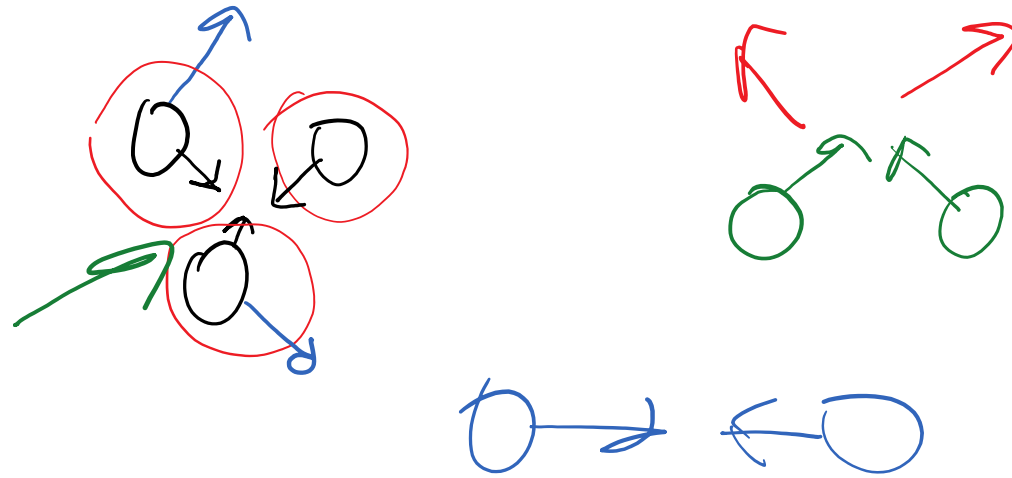
1. Bewegungsgleichungen
2. Objekt aus Linien, in beliebiger Orientierung zeichnen
3. Pos setzen
4. an der Pos in beliebig gedrehter Lage zeichnen

Simulation eines Camera kontrollierten Bertl Roboters

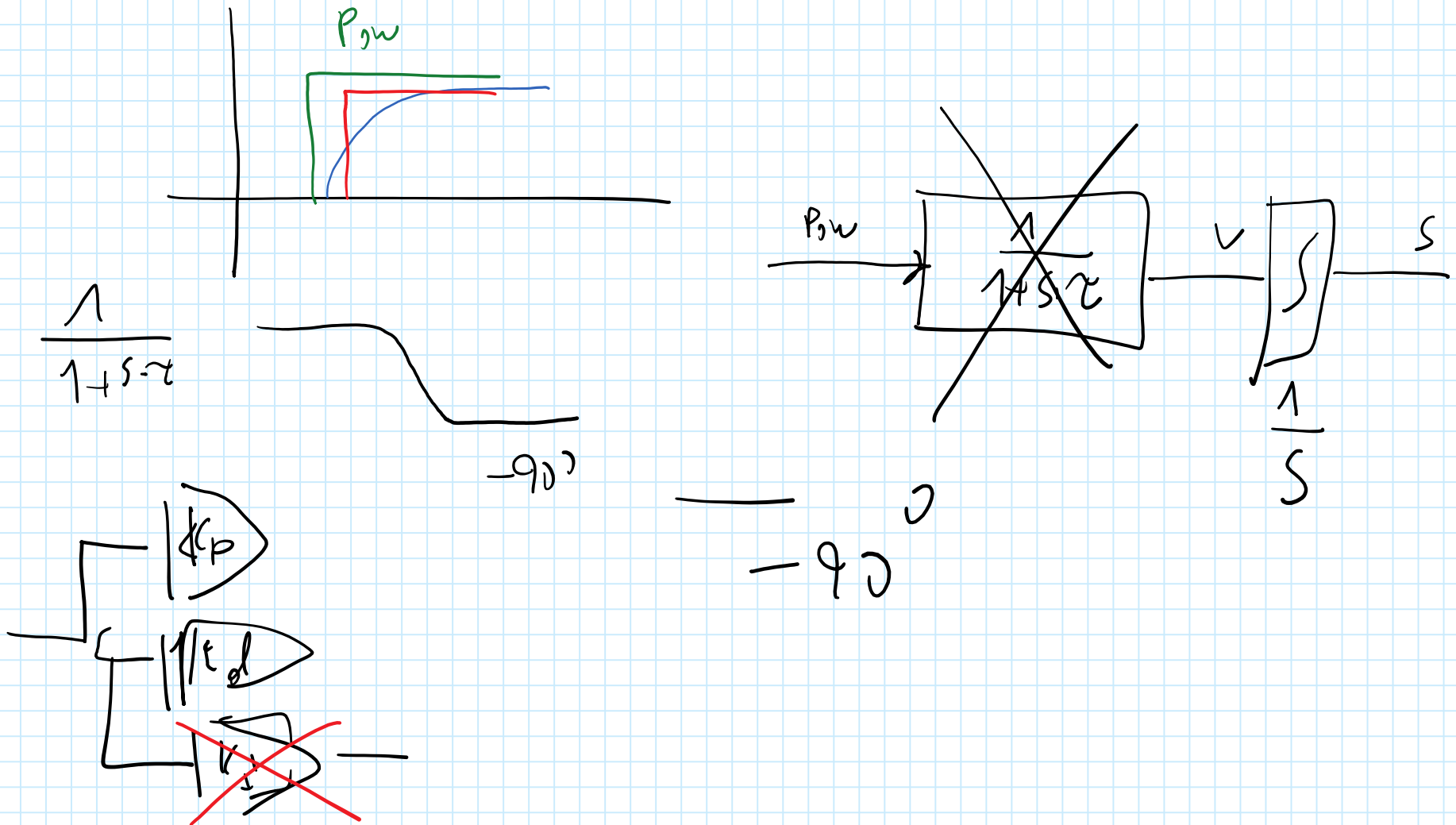
Bewegung Tennisball

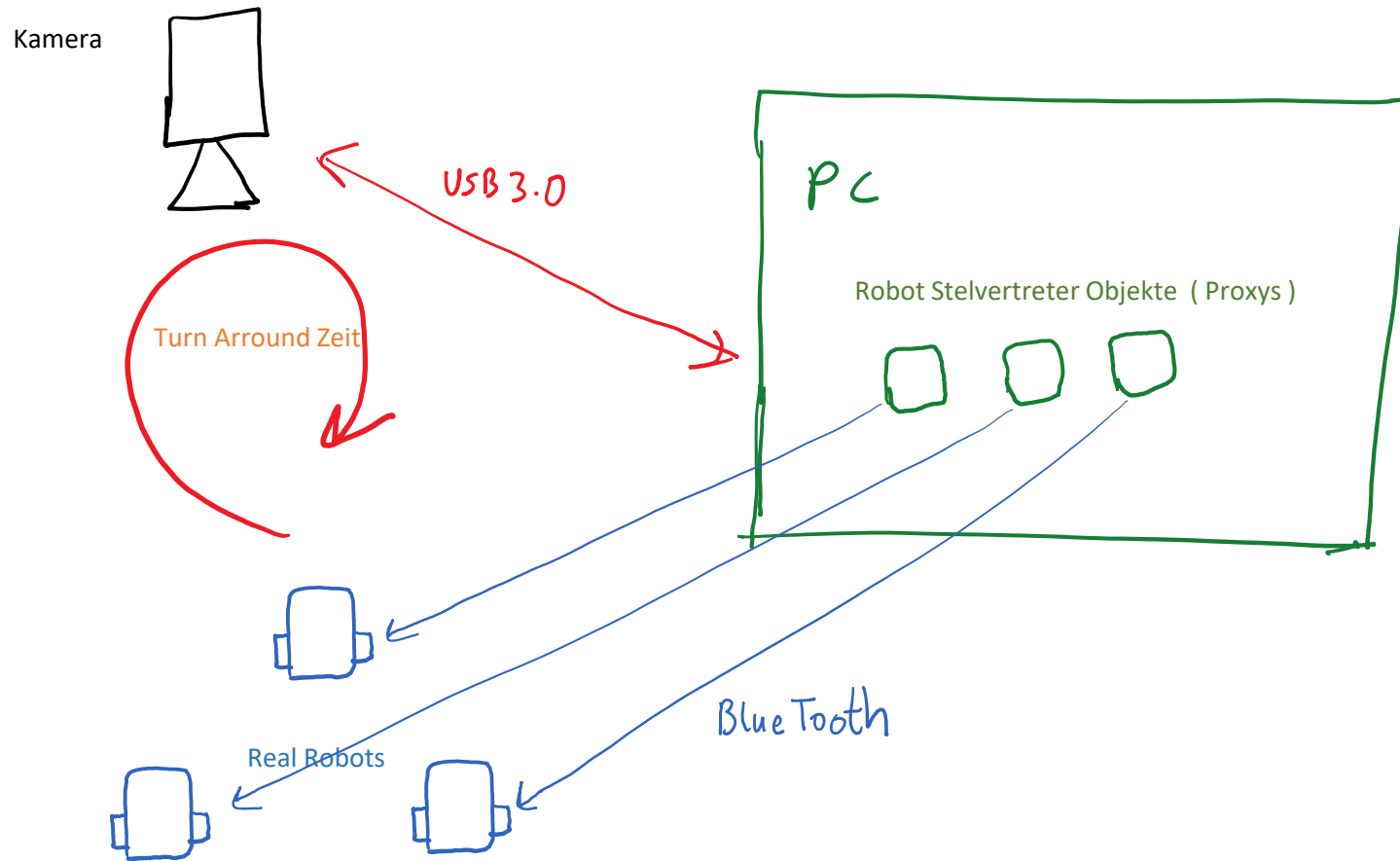
Billard (nichtzenraler Stoß)

Bewegung eines Satelliten um die Erde (Sonne, Galaxis)

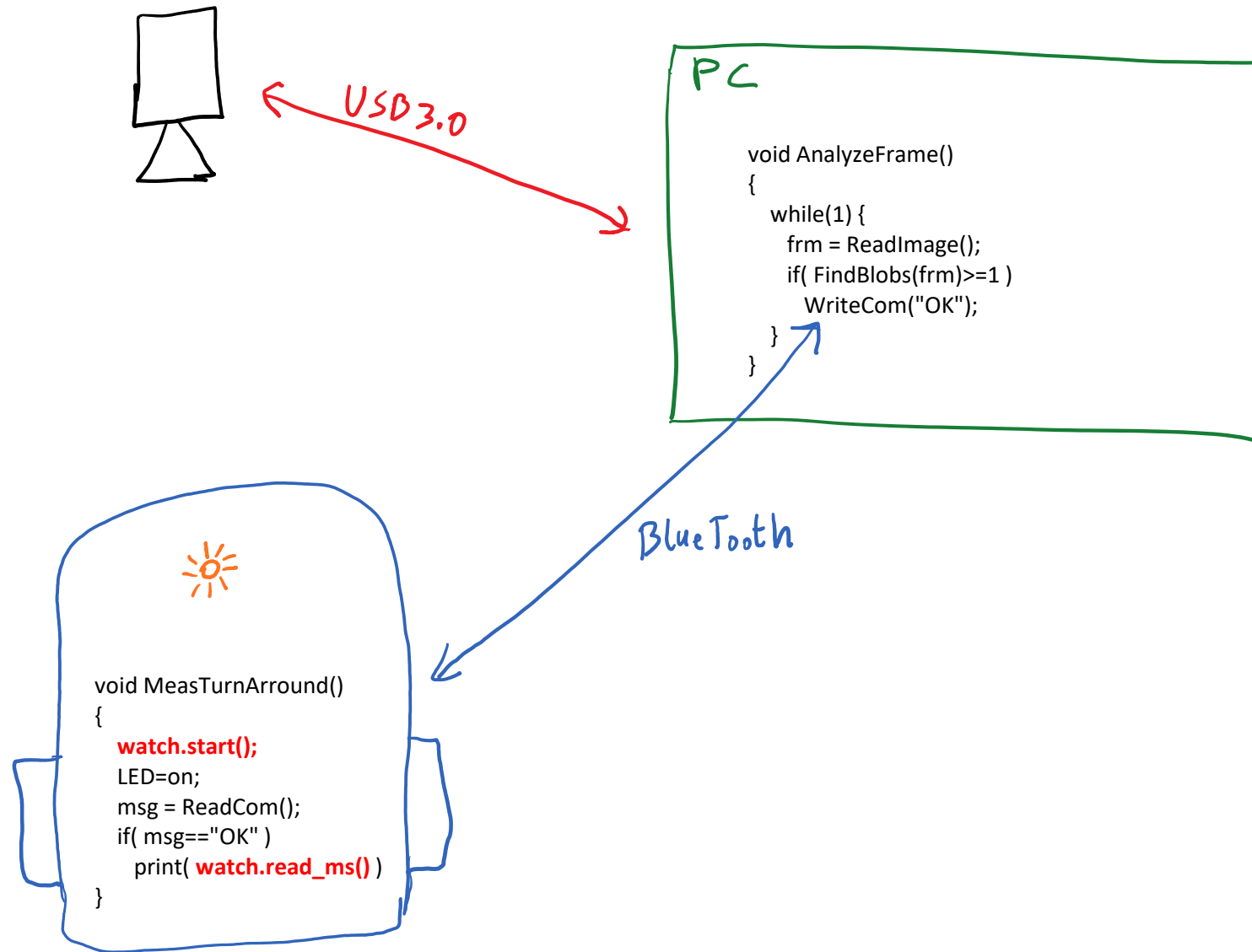


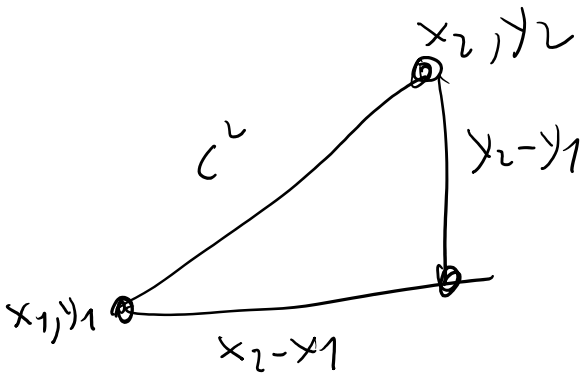
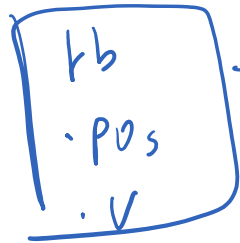
21 Misc



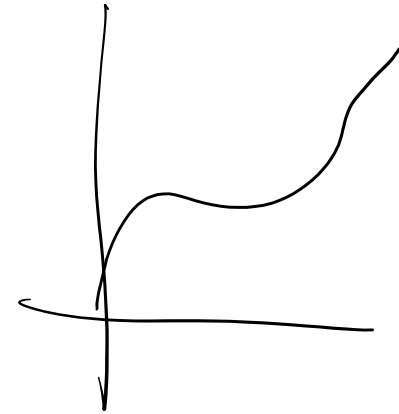
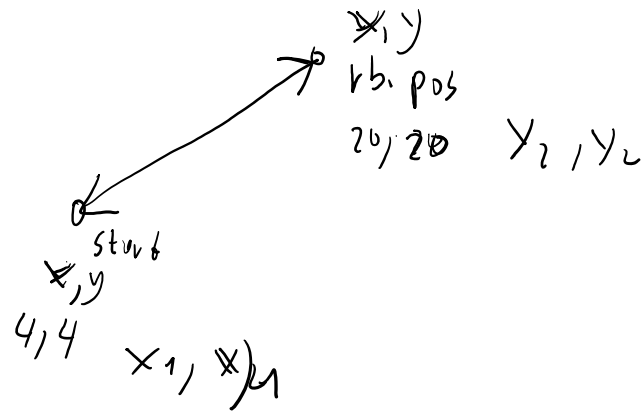


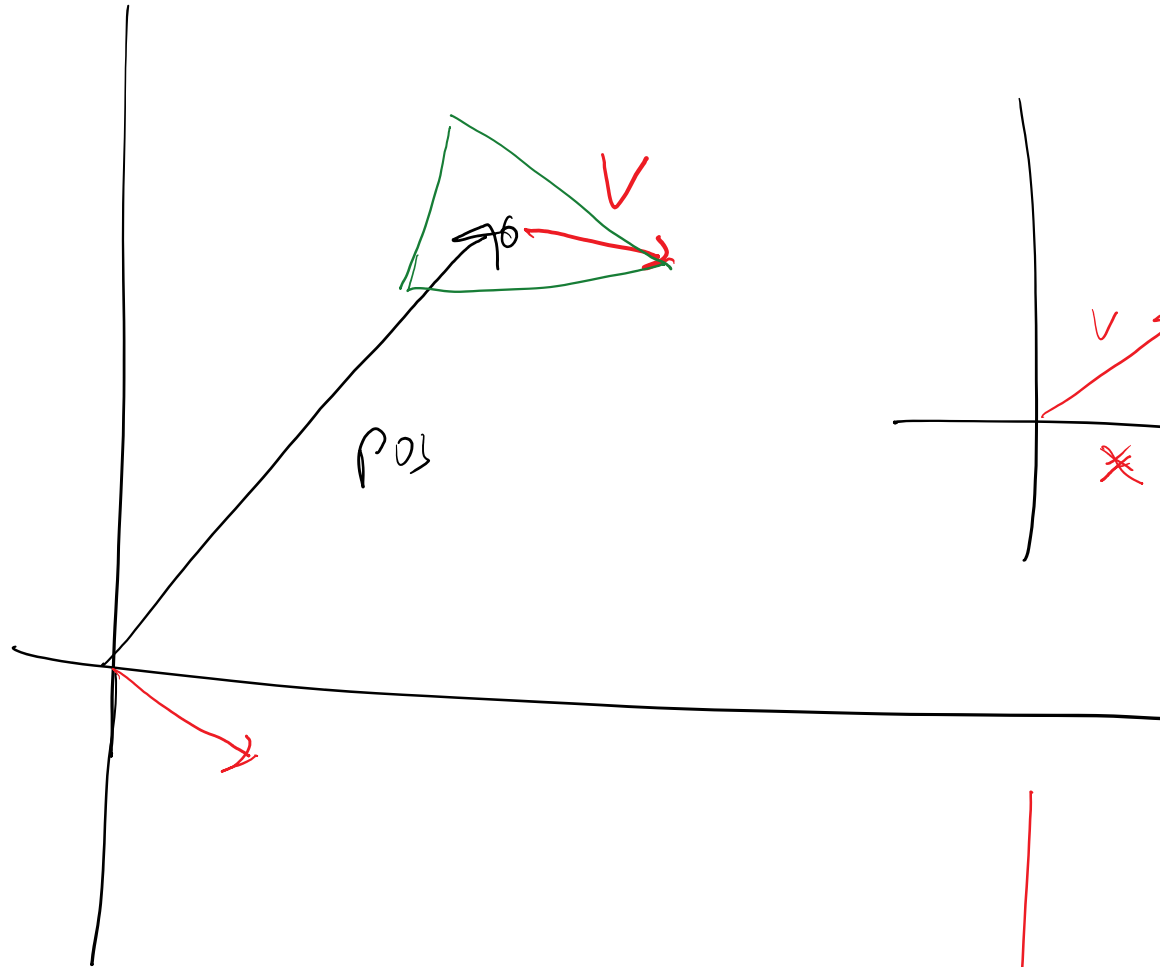
23 Messung der Turn Around Zeit





$$\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$





$$\alpha \tan\left(\frac{y}{x}\right)$$

