SVEUČILIŠTE U ZAGREBU
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 3878

Modeliranje sustava koji iskače iz vode pod utjecajem uzgona

Donat Jakovčev

Zagreb, srpanj 2015.

Umjesto ove stranice umetnite izvornik Vašeg rada.
Da bi ste uklonili ovu stranicu obrišite naredbu `\izvornik`.

SADRŽAJ

1. Uvod

Ljudsku povijest krase brojni pronalasci i izumi koji su čovjeku omogućili prodor u do tad nepristupačne dijelove svijeta pa je tako cijeli kopneni svijet odavno detaljno istražen i kartografiran, ali morske dubine su još uvijek velikom većinom ostale neistražene. Od cjelokupne Zemljine površine morska površina, čiji je svijet pun misterija, zauzima 72%. Zahvaljujući novim tehnologijama podvodnih vozila u mogućnosti smo sve više i dublje istražiti taj misterij. Takva istraživanja uglavnom se odvijaju bez prisutnosti ljudske posade, pomoću autonomnih podnovnih vozila(AUV[1]).

Zadatak ovog završnog rada bio je definirati fizikalni model kuglice koja zaranja do zadane dubine, izranja i iskače iz vode, to jest definirati dinamičko ponašanje kuglice ovisno o sili teži, dinamičkom otporu tekućine i sili uzgona za vrijeme gibanja kroz vodu, zrak i pri prijelazu iz medija u medij. Kuglica je zapravo model sonde koju bi istraživaći ispustili iznad nepoznatih dubina. Na zadanoj dubini sonda povećava plovnost i kreće u izron sa prikupljenim podatcima. Plovnost se može povećati pražnjenjem šuplje komore u kuglici ili otvaranjem zračnog mjehura. Takva kuglica brzo će izranjat i pri izlasku iz mora imati će dovoljno inercije da dostigne visine i do jednog metra što je bitno za uspješnu komunikaciju s matičnim plovilom. Nakon uspješne komunikacije kuglica pušta more u šuplju komoru (ili otpušta mjehur) i ponovno zaranja čime započinje novi ciklus.

Kako bismo odredili primjenjivost ovakvog modela konačno je potrebno simulirati veći broj takvih kuglica u ROS[2][3] razvojnom okruženju čime možemo saznati potreban broj sondi zadanih parametara mase i promjera za rad na zadanoj dubini.

[1] Autonomna podvodna vozila (engl. *Autonomous Underwater Vehicle*)

[2] (engl. *Robot Operating System*)

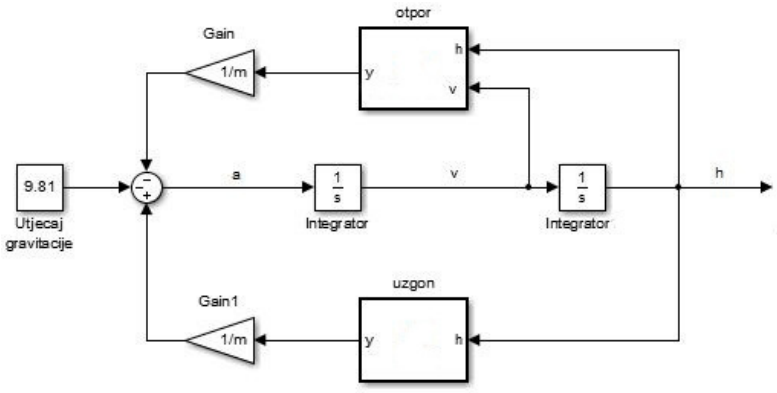
2. Matematički model

Dinamiku kuglice koja izranja opisujemo sa silom teže koja djeluje prema dolje, uzgonom koji djeluje prema gore i silom otpora fluida koja se suprotstavlja kretanju kuglice.

- **Sila uzgona:** $F_u = \rho_f * V_u * g$
  gdje su ρ_f gustoća fluida u kojem se nalazi kuglica, V_u je volumen uronjenog dijela kuglice i mijenja se pri izronu, a g konstanta sile teže i iznosi $9.81\ m/s^2$
- **Sila teže:** $F_g = m * g$
  gdje je m masa kuglice
- **Sila otpora fluida:** $F_d = 0.5 * \rho_f * v^2 * C_D * A$
  gdje su v brzina kuglice, C_D konstanta otpora tijela u fluidu i iznosi 0.47 za sferna tijela, a A je površina kuglice koja se suprotstavlja kretanju

Postavlja se jednadžba: $F = F_u - F_g - F_d$, koja nakon uvrštavanja sila i kratkog premještanja varijabli ima oblik:

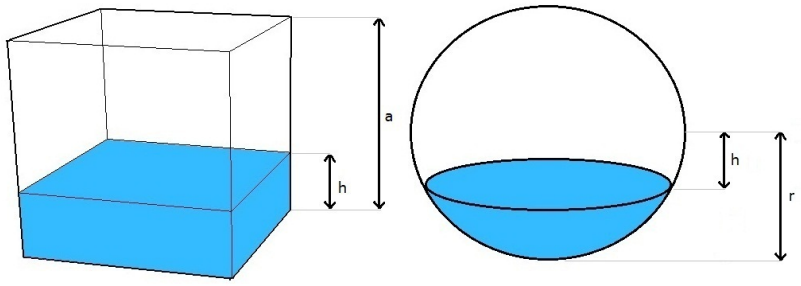
$$\boxed{\ddot{h} = g + \frac{1}{m}(0.5 * \rho_f * C_D * A * \dot{h}^2 - \rho_f * V_u * g} \tag{2.1}$$



Slika 2.1: Shematski prikaz matematičkog modela

2.1. Problem promijene volumena uronjenog dijela tijela

Na samom početku izrade modela usredotočili smo se na druge probleme pa smo umjesto kuglice, radi lakšeg proračuna promjene uronjenog djela tijela na koji djeluje sila uzgona radili model s kockom. Nakon što je simulacija dala pozitivne rezultate prešli smo na model s kuglicom promjenom jednadžbe u bloku koji izračunava uzgon i površine u bloku za računanje otpora fluida. Jednadžbe za uronjeni dio kuglice uvijek vrijedi(za h<r i za h>r).



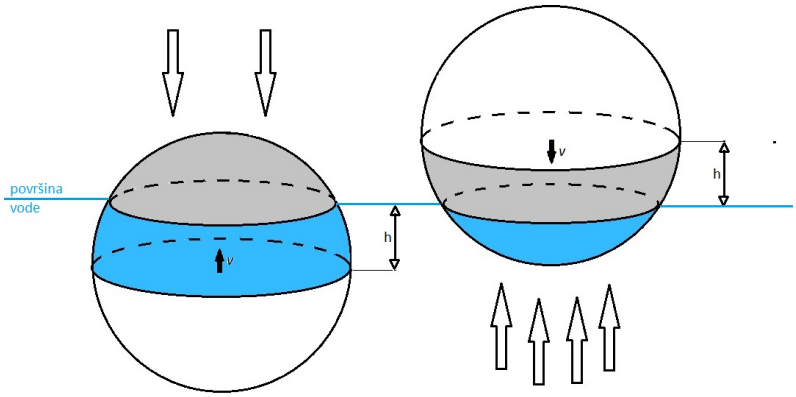
Slika 2.2: Skica uronjenog dijela kocke i kuglice

Jednadžbe za uronjeni dio kuglice uvijek vrijedi(tj. za h<r i za h>r):
Kocka: $V_u = a^2 * h$
Kugla: $V_u = \frac{1}{3} + \pi * (r + h)^2 * (2 * r - h)$

2.2. Problem promijene udarnog oplošja tijela

Površina kocke koja se suprotstavlja kretanju uvijek je jednaka A = a2 pa se sila otpora fluida skokovito mijenja u trenutku kada gornja ploha prođe kroz površinu vode. U izrazu za silu otpora fluida ($F_d = 0.5 * \rho_f * v^2 * C_D * A$) mijenja se, osim brzine, samo ρ_f – gustoća fluida pa slijedi da je otpor fluida u vodi veći tisuću puta nego u zraku.

Kada se kuglica kreće kroz fluid, vodu ili zrak, udarna površina kuglice je prednja polusfera $A = 2 * r^2 * \pi$. Ali kada kuglica prolazi kroz površinu fluida, recimo da izranja ($-r < h < 0$), otpor zraka djeluje na izronjenu kapu polusfere(A), a otpor vode samo na uronjeni pojas polusfere(A_K). Pri uranjanju ($0 < h < r$) otpor vode djeluje na uronjenu kapicu(A), a zraka na suhi pojas polusfere(A_K).



Slika 2.3: Skica udarne površine

Tablica 2.1: naslov

v> 0	h < -r	$F_d = 0.5 * \rho_v * v^2 * C_D * A$
	-r < h < 0	$F_d = 0.5 * v^2 * C_D * (A * \rho_z + (A - A_K) * \rho_v)$
	h > 0	$F_d = 0.5 * \rho_z * v^2 * C_D * A$
v <0	h<0	$F_d = 0.5 * \rho_v * v^2 * C_D * A$
	0<h<r	$F_d = 0.5 * v^2 * C_D * (A * \rho_v + (A - A_K) * \rho_z)$
	h>r	$F_d = 0.5 * \rho_z * v^2 * C_D * A$

3. Rad u ROS-u

3.1. Programiranje u ROS-u

3.1.1. O ROS-u

ROS pruža i omogućava korištenje raznih programskih alata za stvaranje robotskih aplikacija. Također omogućava sklopovsku apstrakciju, vizualizaciju, slanje poruka, upravljanje paketima, nisku razinu upravljanja uređajima, paralelno izvođenje programa, itd. On nije operacijski sustav u tradicionalnom smislu upravljanja i planiranja, već pruža strukturirani komunikacijski sloj. ROS je jezično neutralan kako bi olakšao korištenje sustava što većem broju korisnika. Podržava rad s C++, Pyhton, Octave, Java te LISP, a u razvoju je i podrška za još neke programske jezike. U ROS-u se koriste standardne biblioteke za sve algoritme i uređaje. Građa ROS sustava se provodi modularno unutar stabla izvornog koda. [3]

Programi u ROS-u su strukturirani u pakete, a paket sadrži nodove koji su najmanja izvršna cjelina projekta u ROS-u. Nodovi međusobno komuniciraju tako da node Publisher(hrv. *objavljivač*) objavi veličinu zadanog tipa (npr. int) na Topic-u (hrv. *temi*), dok drugi node, Subscriber(hrv. *pretplatnik*) automatski izvršava kod sa dobivenom veličinom svaki put kad se nešto objavi na temi na koju je pretplaćen. [4]

Parametri u ROS-u funkcioniraju tako da bilo koji node može postaviti vrijednost parametra funkcijom `set_param` ili pročitati vrijednost funkcijom `get_param`. Pri pozivanju `set_param` i `get_param` funkcija treba paziti da parametri mogu pripadati paketu ili određenom nodu jer se pozivi tih funkcija tada razlikuju. [5]

Za izradu ovog projekta nodove sam programirao u *Python* [2] programskom jeziku, a za simulaciju poslužio sam se *pyGame*[1] paketom.

3.1.2. Ideja

Programiranje u ROS-u je zapravo programiranje u *Python-u* uz poznavanje strukture i uz pridržavanje pravilima ROS-a. Simulacija kuglica je realizirana pomoću dva noda. Prvi nod, simulator, izračunava novu poziciju svake kuglice dok drugi nod, display, uzastopno crta nove pozicije kuglica čime postiže efekt animacije. *Rosparam* klasom ujednačavaju se svi parametri potrebni za rad oba noda, npr. zadana dubina.

3.2. Programiranje noda *simulator*

Da bismo isprogramirali matematički model kuglice opisan u 2. poglavlju najbolje je definirati funkciju za računanje svake sile.

3.2.1. Računanje diferencijske jednadžbe

1. Funkcija za računanje sile uzgona jednostavno se napiše u *Python* sintaksi. Ulazni argumenti su dubina na kojoj se kuglica nalazi, radius kuglice, π, ρ_{vode}, ρ_{zraka} i g. Izrazi za računanje uzgona razlikuju se ako je kuglica u vodi, zraku ili na prijelazu iz jednog u drugo, jednostavno jer je u izrazu gustoća fluida u kojem se tijelo nalazi što je već objašnjeno u 2. poglavlju.

```
def uzgon(h,r,pi,rv,rz,g):
    Vu=pi*(r+h)^2*(2*r-h)/3                    #uronjeni dio
    V=4*r^3*pi/3                               #cijeli volumen
    if h < -r:                                 #kuglica je u vodi
        y=rv*g*V                               #izraz za uzgon
    elif h > r:                                #kuglica je u zraku
        y=rz*g*V                               #izraz za uzgon
    else:                          #kuglica je na prijelazu medija
        y=rz*g*Vu+rv*g*(V-Vu)                              #uzgon
    return y
```

2. Funkcija za računanje sile dinamičkog otpora fluida ima nešto više koda, ali i dalje je jednostavna, a duža je samo zato što ima duplo više slučajeva koje treba razmotriti. Ulazni parametri su h, v, r, ρ_{vode}, ρ_{zraka} i C_D. Pri analizi treba obratiti pozornost na to da nas zanima samo prednja polukugla i njen položaj u odnosu na površinu.

```
def otpor(h,v,r,rv,rz,d):
    A=2*r^2*pi
    a=2*r*pi*(r-abs(h))
    if v > 0:                              #kuglica zaranja
        if h < -r:                        #cijela je zaronila
            y=0.5*rv*d*A*v^2*predz(v)
        elif h > 0:                        #cijela je u zraku
            y=0.5*rz*d*A*v^2*predz(v)
        else:                            #kuglica ulazi u vodu
            y1=0.5*rz*d*a*v^2*predz(v)
            y2=0.5*rv*d*(A-a)*v^2*predz(v)
            y=y1+y2
    elif h > r:                           #kuglica je u zraku
        y=0.5*rz*d*A*v^2*predz(v)
    elif h < 0:                            #kuglica je u vodi
        y=0.5*rv*d*A*v^2*predz(v)
    else:                                     #kuglica uranja
        y1=0.5*rv*d*a*v^2*predz(v)
        y2=0.5*rz*d*(A-a)*v^2*predz(v)
        y=y1+y2
    return y
```

3. Funkcija *pozicija* samo poziva sve funkcije sila. Ona je ta koja predstavlja matematički model, koja računa jednadžbu diferencija: $h_2 = \frac{T^2}{m} * (F_u - F_d - F_g) + 2 * h_1 - h_0$. Parametar potreban za rad funkcije je osim dosad već spomenutih, h0, v0, h1, v1, r, pi, rv, rz, g ,d, m, je period diskretizacije T.

```
def pozicija(m,h0,h1,v0,v1,T,r,pi,rv,rz,g,d):

    h2=(T^2/m)*
    (uzgon(h1,r,pi,rv,rz,g)-otpor(h1,v1,r,rv,rz,d)-m*g)
    +(2*h1)-h0

    v2=1/T*(h2-h1)
    return (h2,v2)
```

3.2.2. Očitavanje potrebnih parametara

Pri pokretanju programa korisnik može postaviti parametre (zadana dubina, radijus, masa i broj kuglica) ili samo pokrenuti sa zadanim vrijednostima. U nodu *simulator* potrebno je očitati vrijednosti svih tih parametara kako bi se simulacija mogla točno izvesti.

Varijabli s lijeva pridružuje se vrijednost parametra $'x'$, a ako isti nije postavljen, onda varijable ima vrijednost $default$(vrijednost u zagradi iza zareza). Pa će tako na primjer r biti jednak vrijednosti parametra *radius*, ili 0.15 ako korisnik nije promijenio postavljenu vrijednost parametra radius kad je pokrenuo program.

```
r = rospy.get_param('radius', 0.15)
broj_kuglica = rospy.get_param('broj', 5)
h0 = rospy.get_param('zad_dub', -10)
masa = rospy.get_param('masa', 0.5)
```

3.2.3. Rad u zatvorenoj petlji

Simulator u svakom ciklusu izvođenja objavljuje dubine svih kuglica kao niz on *Br_kug* članova. Pošto ne znamo u naprijed s koliko kuglica ćemo raditi jedini podatkovni oblik koji nam paše je Float32MulitArray koji se reinicijalizira na početku ciklusa te mu se slijedno dodaju pojedine dubine kuglica kao float broj. Prije petlje još je potrebno inicijalizirati *publisher* varijablu pri čemu se odabire željeni *topic*, $pozicija$. Slijedi inicijaliziranja svih potrebnih matrica, a te su: matrice posljednjih dva stanja dubine, odnosno brzine svih kuglica, matrica promjene plovnosti i lista masa svih kuglica. Program ulazi u zatvorenu petlju. U svakom prolasku:

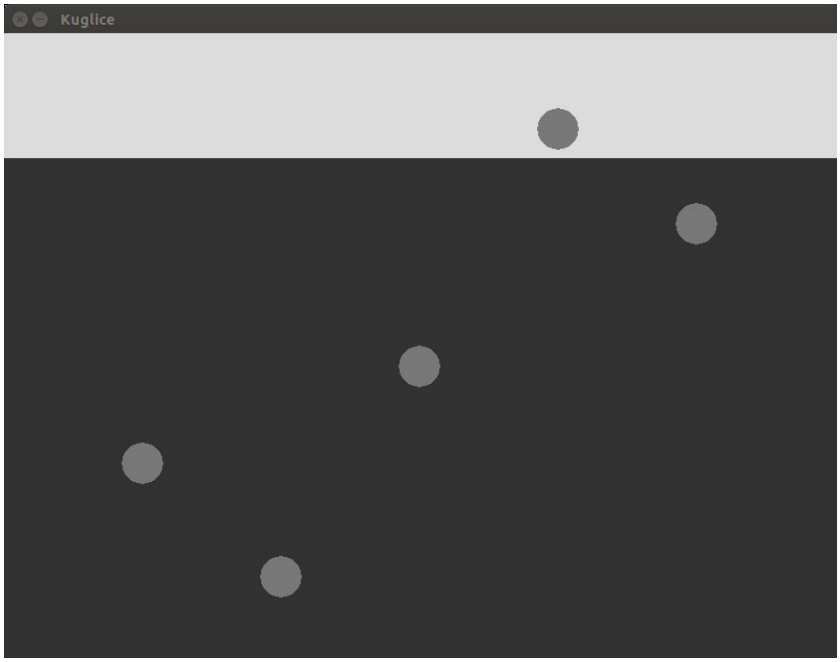
1. U petlji izračunava poziciju svake kuglice i dodaje istu listi
2. Popunjena lista se objavljuje(sadrži dubine svih kuglica za tekući ciklus)
3. Obavlja zamjenu varijabli da bi se ciklus mogao nastaviti ($h_{i-2} = h_{i-1}, ...$)
4. Provjerava dali je kuglica na vrhu ili na dnu i po potrebi mijenja plovnost

3.3. Programiranje noda *display*

Zadatak display noda je osigurati uzastopno crtanje određenog broja kuglica u prozoru zadanih dimenzija.

Nakon registracije noda u ROSovom okruženju, da bi node dobio trenutne dubine kuglica potrebno ga je preplatiti na onu istu temu na kojoj node *Simulator* objavljuje izračunate dubine. *Display* je isprogramiran tako da pri svakoj objavi na temi '*pozicija*' pokreće funkciju koja obradi dubine te ih iscrta u simulacijskom prozoru. Prije nego se počne pozivati ta funkcija, za crtanje, potrebno je funkcijom *get_param*, slično kao u nodu *Simulator*, očitati vrijednost parametra zadane dubine do koje kuglica zaranja. Zadana dubina potrebna je funkciji za crtanje kako bi mogla iz odnosa trenutne i zadane(koja je ujedno najdublja) odrediti poziciju kuglice na ekranu. Nakon očitanja parametra node ne radi ništa, to jest čeka objavu na pretplaćenoj temi.

U svakom pozivu funkcije za crtanje izračunate dubine se preračunavaju u pozicije na ekranu, a zatim se pozadina i kuglice crtaju na novom listu koji se aktivira na kraju poziva ove funkcije.



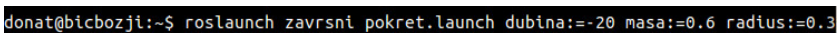
Slika 3.1: Simulacijski prozor

Na slici 3.1 vidimo rad simulatora za 5 kuglica sa zadanom dubinom od $10\ m$, a ovako izgleda podatak koji je node *simulator* poslao nodu *display* u trenutku na slici:

```
data: [-6.989285945892,-9.593377113342,
    -4.780272006988, 0.651706874370, -1.5167967081069]
```

3.4. Pokretanje simulacije

Ovako napravljen paketa u ROS-u jednostavno se pokreće pokretanjem svakog noda zasebno naredbom *rosrun*. Mana pokretanja na ovaj način jest što parametri simulacije(broj kuglica, njihova masa i radius i radna dubina) ne mogu zadati pri pokretanju simulacije. To jest, vrijednosti tih varijabla su one koje zadamo u programu noda prije pokretanja Ali ako varijable povežemo s ROS parametrima onda možemo napisati *launch* program $pokret.launch$ koji će pokrenut oba noda i učitati nove parametre pri pokretanju. Cijeli kod $pokret.launch$ programa nalazi se u dodatku 5.3.



Slika 3.2: Primjer pokretana simulacije

4. Zaključak

ROS se pokazao kao praktičan program i jako dobar alat za simulaciju. Ne samo za crtanje, kako je to ovdje realizirano, nego i za ozbiljne 3D simulacije robota. Koncept ROS-a je neuobičajan i zato treba malo vremena da ga se upozna.

Ova tema bi se mogla još puno istraživati, a sam završni rad bi mogao poslužiti kao okruženje za napredniju simulaciju. U njoj bi postojao nod za upravljanje za svake kuglice (jer bi se tako u stvarnosti morao realizirati) u kojem bi se, poznavajuću trenutnu dubinu, mijenjala plovnost i primale odnosno odašiljale poruke, a node *simulator* bi u takvoj simulaciji poslužio kao matematička pozadina da zamjeni dubinometar i "kaže" kuglici na kojoj je dubini.

Tako robotizirane kuglice koje bi imale ugrađenu kameru, mogle bi se na primjer isprogramirati da prate kabel i cijeloga ga slikaju, kako bi se što prije pronašao kvar na kabelu.

Uz kvalitetnu obradu slike i uz pomoć kemijskih senzora jato kuglica moglo bi relativno brzo analizirati stupanj zagađenja mora i napraviti trodimenzionalnu sliku izljevene nafte ili neke opasne kemikalije što bi pomoglo u sanaciji zagađenja.

Kuglice mogu poslužiti za daljinsko upravljanje ronilice na dubini. Mogu nositi podatke o zadanom smjeru kretanja s istraživačkog broda na ronilicu, a vraćati poziciju robota i sliku s kamere. Tada bi se kuglice dodatno isprogramirale da prate smjer kretanja robota.

Sva navedena unaprjeđenja traže od kuglica mogućnost kretanja po *x*-*y* ravnini što bi se moglo postići ugradnom 4 bočne peraje. Takav sustav kuglica sigurno bi se mogao nastaviti istraživati i, u budućnosti, pronaći pravu primjenu. Vjerojatno više njih.

LITERATURA

[1] Pygame wiki. URL https://www.pygame.org/wiki/about.

[2] Python. URL https://www.python.org/about/.

[3] Ros wiki. URL http://www.ros.org/wiki/.

[4] Aaron Martinez i Enrique Fernández. *Learning ROS for Robotics Programming*. Packt Publishing, Rujan 2013. ISBN 1782161449. URL http://www.packtpub.com/learning-ros-for-robotics-programming/book.

[5] Jason M. O'Kane. *A Gentle Introduction to ROS*. CreateSpace Independent Publishing Platform, Listopad 2013. ISBN 9781492143239. URL http://www.cse.sc.edu/~jokane/agitr/.

Modeliranje sustava koji iskače iz vode pod utjecajem uzgona

Sažetak

U završnom radu definiran je fizikalni model kuglice koja zaranja do zadane dubine, izranja i iskače iz vode. Definirano je dinamičko ponašanje takve kuglice u ovisnosti o sili teži, sili dinamičkog otpora fluida i sili uzgona za vrijeme gibanja kuglice kroz vodu, zrak i na prijelazu medija. Poanta iskakanja je u ostvarivanju bežične komunikacije s plovilom u blizini i prijenosom prikupljenih podataka u prošlom zaronu. Jer kuglica je zapravo model sonde koja se može nadograditi kamerom ili senzorima za kemijsku analizu vode i tako istražiti podmorje.

Ključne riječi: Ključne riječi, odvojene zarezima. ROS, podvodna komunikacija, simulacija modela, istraživanje podmorja, daljinsko upravljanje

Modelling of an buoyancy-based aquatic jumping system

Abstract

This thesis quest was to define realistic physical model of a ball which is sinking to given depth, emerging and jumping above water. To define dynamics of such ball in relation to force of graviti, drag force and buyoince force while going through water, air and upon crossing between. The point of jumping above water is in establishing wireless connection with main vessel nearby and thus transfer collected data from the last dive. Be cause ball in model of a probe which is upgradable with camera or chemical probes and such enables thorough underwater exploration.

Keywords: Keywords. ROS, underwater communication, simulation of model, underwater exploring, remote control

5. Dodatak

5.1. Kod noda *simulator.py*

```
#!/usr/bin/env python
import rospy
from std_msgs.msg import Float32MultiArray

pi = 3.14
rv = 1000
rz = 1.225
g = 9.81
d = 0.47
T = 0.001

def uzgon(h,r,pi,rv,rz,g):
    Vu=pi*(r+h)^2*(2*r-h)/3                    #uronjeni dio
    V=4*r^3*pi/3                               #cijeli volumen
    if h < -r:                                 #kuglica je u vodi
        y=rv*g*V                               #izraz za uzgon
    elif h > r:                                #kuglica je u zraku
        y=rz*g*V                               #izraz za uzgon
    else:                                      #kuglica je na prijelazu medija
        y=rz*g*Vu+rv*g*(V-Vu)                  #uzgon
    return y
```

```
def otpor(h,v,r,rv,rz,d):
    A=2*r^2*pi
    a=2*r*pi*(r-abs(h))
    if v > 0:                              #kuglica zaranja
        if h < -r:                        #cijela je zaronila
            y=0.5*rv*d*A*v^2*predz(v)
        elif h > 0:                        #cijela je u zraku
            y=0.5*rz*d*A*v^2*predz(v)
        else:                            #kuglica ulazi u vodu
            y1=0.5*rz*d*a*v^2*predz(v)
            y2=0.5*rv*d*(A-a)*v^2*predz(v)
            y=y1+y2
    elif h > r:                           #kuglica je u zraku
        y=0.5*rz*d*A*v^2*predz(v)
    elif h < 0:                            #kuglica je u vodi
        y=0.5*rv*d*A*v^2*predz(v)
    else:                                    #kuglica uranja
        y1=0.5*rv*d*a*v^2*predz(v)
        y2=0.5*rz*d*(A-a)*v^2*predz(v)
        y=y1+y2
    return y

                #funkcija za izracunavanje novu dubinu
def pozicija(m,h0,h1,v0,v1,T,r,pi,rv,rz,g,d):
  h2=(T^2/m)*
  (uzgon(h1,r,pi,rv,rz,g)-otpor(h1,v1,r,rv,rz,d)-m*g)+
  (2*h1)-h0
  v2=1/T*(h2-h1)
  return (h2,v2)

                                        #glavna funkcija
def simulator(br_kug,h0,pi,rv,rz,g,T,d):
    h=[[0 for x in range(3)] for x in range(br_kug)]
    v=[[0 for x in range(3)] for x in range(br_kug)]
    ch=[0 for x in range(br_kug)]
    temp=[0 for x in range(br_kug)]
```

```
    vozi=[0 for x in range(br_kug)]
m=[masa for x in range(br_kug)]

pub=rospy.Publisher('pozicija',Float32MultiArray,
queue_size=10)                    #registracija topica
rate=rospy.Rate(1/T)                              #1000hz
vozi[0]=1                          #prva kuglica krece
for i in range(br_kug):       #kuglica tone na pocetku
  temp[i]=m[i]
  ch[i]=1
  m[i]=3*4188.79*r**3

while not rospy.is_shutdown():
  dubina=Float32MultiArray()              #prazan niz
  for i in range(br_kug):
    if vozi[i]==1:
      (h[i][2],v[i][2])=
        pozicija(m[i],h[i][0],h[i][1],v[i][0],
        v[i][1],T,r,pi,rv,rz,g,d)  #racunanje pozicije
    dubina.data.append(h[i][2]) #dodavanje nove dubine
    h[i][0]=h[i][1]         #priprema za sljedeci ciklus
    h[i][1]=h[i][2]
    v[i][0]=v[i][1]
    v[i][1]=v[i][2]
                              #promjena plovnosti na vrhu
    if v[i][2]<0 and ch[i]==0 and vozi[i]==1:
      temp[i]=m[i]
      ch[i]=1
      m[i]=3*4188.79*r**3
                              #promjena plovnosti na dnu
    elif h[i][2]<h0 and ch[i]==1:
      m[i]=temp[i]
      ch[i]=0
                             #pokretanje sljedece kuglice
    if h[i][2]<(2*h0/(br_kug+1)):
      if i+1<br_kug:
```

```
            vozi[i+1]=1
    pub.publish(dubina) #objava novih dubina svih kuglica
    rate.sleep()

if __name__ == '__main__':
        #inicijalizacija noda i citanje ros parametara
    rospy.init_node('simulator', anonymous=True)
    r = rospy.get_param('radius', 0.15)
    broj = rospy.get_param('broj', 5)
    h0 = rospy.get_param('zad_dub', -10)
    masa = rospy.get_param('masa', -10)
    try:
        simulator(broj, h0, pi, rv, rz, g, T, d)
    except rospy.ROSInterruptException:
            pass
```

5.2. Kod noda *display.py*

```
#!/usr/bin/env python
import pygame
import rospy
from std_msgs.msg import Float32MultiArray

pygame.init()
disp_sirina = 800
disp_visina = 600
blue = (0, 0, 255)
sky_blue = (135, 206, 250)
red = (255, 0, 0)

gameDisplay =                       #prozor za simulaciju
  pygame.display.set_mode((disp_sirina, disp_visina))
pygame.display.set_caption('Kuglice')


def callback(data):
```

```
    h0 = rospy.get_param('zad_dub', -10)
    r = rospy.get_param('radius', 0.15)

    velicina = 20   #postavljanje parametara simulacije
    br_kug = len(data.data)
    dubina = [0 for x in range(br_kug)]
    pozicija_x = [0 for x in range(br_kug)]
    pozicija_y = [0 for x in range(br_kug)]

    gameDisplay.fill(blue)
    pygame.draw.rect(gameDisplay, sky_blue,
      [0, 0, disp_sirina, disp_visina / 5])

  for i in range(br_kug):
      dubina[i] = data.data[i]          #citanje dubine
      pozicija_y[i]=int(disp_visina/5+     #racunanje y
        (14*dubina[i]*disp_visina/(20*h0)))       # i x
      pozicija_x[i] = int(disp_sirina/(br_kug+1)*(i+1))
      pygame.draw.circle(gameDisplay,red,      #crtanje
        [pozicija_x[i],pozicija_y[i]],velicina)
    pygame.display.flip() #sve nove kuglice idu na ekran


def listener():
    rospy.init_node('listener', anonymous=True)
    rospy.Subscriber("pozicija", Float32MultiArray,
       callback, queue_size = 1)   #pretplata na topic
    rospy.spin()

if __name__ == '__main__':
    listener()
```

5.3. Kod pokterača *pokret.launch*

```
<launch>
  <arg name="dubina" default="-10"/>
  <arg name="masa" default="0.5"/>
  <arg name="r" default="0.15"/>
  <arg name="broj" default="5"/>
```

•Inicijalizaija dodatnih parametara pri pokretanju i definiranje zadane vrijednosti

```
  <rosparam param="broj" subst_value="True">$(arg broj)
  </rosparam>
  <rosparam param="dub" subst_value="True">$(arg dubina)
  </rosparam>
  <rosparam param="masa" subst_value="True">$(arg masa)
  </rosparam>
  <rosparam param="radius" subst_value="True">$(arg r)
  </rosparam>
```

•Postavljanje parametara u rosu na vrijednosti pri pokretanju

```
  <node pkg="zavrsni" type="sim.py" name="Simulator"/>
  <node pkg="zavrsni" type="disp.py" name="Display" />
```

•Pokretanje nodova

```
</launch>
```