Relazione

Progetto A.A. 2022/2023
ADAS made trivial: rappresentazione ispirata alle interazioni in un sistema di guida autonoma

Diciotti Matteo	Manucci Agostino	Montes Anacona Àlvaro
7072181	7084379	7117731

1 giugno 2023 - 22 giugno 2023

Titolo Progetto A.A. 2022/2023 – ADAS made trivial: rappresentazione ispirata alle interazioni in un sistema di guida autonoma

Autori Lista degli autori ordinata per numero di matricola

Matricola	Cognome	Nome	e-mail
7072181	Diciotti	Matteo	matteo.diciotti@stud.unifi.it
7084379	Mannucci	Agostino	agostino.mannucci@stud.unifi.it
7117731	Montes Anacona	Álvaro	alvaro.montes@stud.unifi.it

Obbiettivo Obiettivo del progetto è costruire un'architettura, estremamente stilizzata e rivisitata, per sistemi ADAS, descrivendo possibili interazioni e alcuni comportamenti tra componenti in scenari specifici.

Introduzione

Introduzione al progetto Un ***A**dvanced **D**river **A**ssistance **S**ystem* è un sistema composto da varie componenti che cooperano affinché un veicolo o un mezzo possano assistere un conducente nella guida e in alcuni contesti sostituirsi al guidatore stesso.
Il progetto realizzato cerca di riprodurre fedelmente il sistema simulativo di un ADAS presentato nella richiesta[1] dell'elaborato nella quale sono definite varie componenti suddivise in quattro gruppi: *interfaccia*, *attuatori*, *sensori* e *controllo*.
La simulazione è resa effettiva dall'inserimento di stringhe rappresentanti le azioni che dovrebbero essere eseguite dalle varie componenti di un ADAS in dei file di log specifici per ogni componente. In particolare le componenti implementate sono nove, sette obbligatorie e due facoltative[2].
I *sensori*, attraverso l'acquisizione di dati da file predefiniti o da sorgenti casuali, simulano l'acquisizione

[1] Per visionare la richiesta dell'elaborato riferirsi al documento Allegato_1.pdf compreso nella cartella del progetto
[2] Per conoscere quali elementi facoltativi sono stati implementati visionare la tabella

di dati da parte di sensori reali e li inviano all'unica componente di controllo, la componente *Central ECU*. Gli *attuatori* ricevono messaggi dall'unità di controllo ed eseguono scritture nell'apposito file di log per simulare la messa in atto dell'attuazione del comando su di un attuatore reale. Infine la componente con cui si interfaccia l'esecutore del programma è la *Human-Machine Interface* che simula l'interfaccia del guidatore attendendo comandi (in questo caso scritti su di un terminale) e mostrando i risultati e le conseguenze dei propri comandi (nel caso della simulazione saranno scritti su di un differente terminale tutti i comandi che l'unita di controllo centrale ha inviato alle varie componenti).

Impostazione del lavoro Al fine di realizzare un sistema che simulasse un ADAS sono state eseguite alcune fasi per il completamento del programma: l'analisi delle richieste, la strutturazione teorica dell'architettura del sistema, l'implementazione pratica, la verifica e revisione dell'elaborato.
Ad una prima fase di studio delle richieste congiunta tra gli autori del progetto è immediatamente succeduta la fase di strutturazione dello stesso, anch'essa svolta unitamente tra i relatori, giungendo alla teorizzazione dell'architettura mostrata in figura 1. Sono stati quindi divisi i lavori per l'implementazione del programma tra i componenti del gruppo e uniti gli elaborati in un unico componimento comprensivo di makefile e file binari per l'esecuzione in modalità *ARTIFICIALE* del programma. Sono state quindi eseguite ciclicamente le due fasi di verifica e di revisione: nella prima fase sono stati svolti test per verificare il corretto funzionamento dell'elaborato per entrambe le modalità e nella seconda fase sono state apportate le modifiche necessarie affinché il programma producesse i risultati attesi. Infine e' stata prodotta la presente relazione.

Specifiche

Caratteristiche HW e SW Per l'implementazione sono stati utilizzati tre hardware differenti con tre sistemi operativi differenti:

- **ArchLinux**, architettura x86_64, kernel Linux 6.1.34-1-LTS
- **Manjaro 23.0.0**, architettura x86_64, kernel Linux 5.15.114-2-MANJARO
- **Ubuntu 20.04**, architettura x86_64, kernel Linux 5.15.0-75-generic

Il progetto è stato ideato avendo come obbiettivo un programma portatile che fosse preimpostato per eseguire su di un sistema **Ubuntu Linux 22.04**[3].

Istruzioni compilazione ed esecuzione Nel presente paragrafo si mostrano i passaggi necessari affinche si possa eseguire il programma sul proprio dispositivo:

Estrazione e compilazione

1. Estrarre il file .zip in una cartella a piacimento. (Esempi comando: `tar -x $PROJECT_NAME$;` `7z x $PROJECT_NAME$`);
2. Spostarsi nella cartella creatasi con l'estrazione. (Esempio comando: `cd ./$PROJECT_NAME_FOLDER$`);
3. Eseguire il comando di installazione e la compilazione del programma `make install`.

Terminata questa procedura il programma dovrebbe risultare pronto per l'esecuzione.

[3]NOTA SULLA PORTABILITÀ: nel programma è stata utilizzata di frequente la funzione `signal(2)`, il cui utilizzo è sconsigliato per motivi di portabilità dal manuale a favore della funzione `sigaction(2)`. Dato che il progetto è sviluppato per fini strettamente accademici, come autori abbiamo preferito non distaccarsi dalle nozioni presentate durante le lezioni sottolineando però che per una maggiore portabilità del sistema sarebbe richiesta una variazione nel suddetto contesto.

Esecuzione

1. Assicurarsi di essere con un terminale nella directory sorgente del programma, la stessa contenente le cartelle bin, src, include etc. e il file eseguibile ADAS-simulator.

2. Avviare l'inizializzazione e l'esecuzione del programma tramite la chiamata di `./ADAS-simulator $MODALITÀ$`, dove `$MODALITÀ$` rappresenta la modalità `ARTIFICIALE` o `NORMALE` del programma[4]. Oltre al secondo argomento (il quale risulta obbligatorio, pena la mancata esecuzione del programma e la restituzione di un errore), il comando supporta anche l'opzione `--term` nella quale si può specificare l'emulatore di terminale che si desidera visualizzare come terminale di output. L'emulatore impostato di default è gnome-teminal.

3. Eseguito il comando dovrebbe aprirsi automaticamente un nuovo terminale rappresentante il terminale di output. Non eseguire alcun comando sul terminale di output e posizionarsi sul terminale di input, ovvero il terminale nel quale è stato eseguito il comando di avvio del programma.

4. Il programma è inizializzato, si possono quindi eseguire i comandi INIZIO, PARCHEGGIO per iniziare l'esecuzione del programma e successivamente ARRESTO per simulare un arresto del veicolo[5].

Elementi facoltativi La tabella a pagina 4 mostra gli elementi facoltativi implementati seguiti da una breve descrizione dell'implementazione.

[4]Per la comprensione delle differenze delle modalita di esecuzione riferirsi al paragrafo 4 della richiesta *Allegato_01.pdf*
[5]Per la comprensione dei comandi inseribili dal terminale rifarsi al paragrafo 2, sotto-paragrafo *Human-Machine Interface*, della richiesta *Allegato_01.pdf*

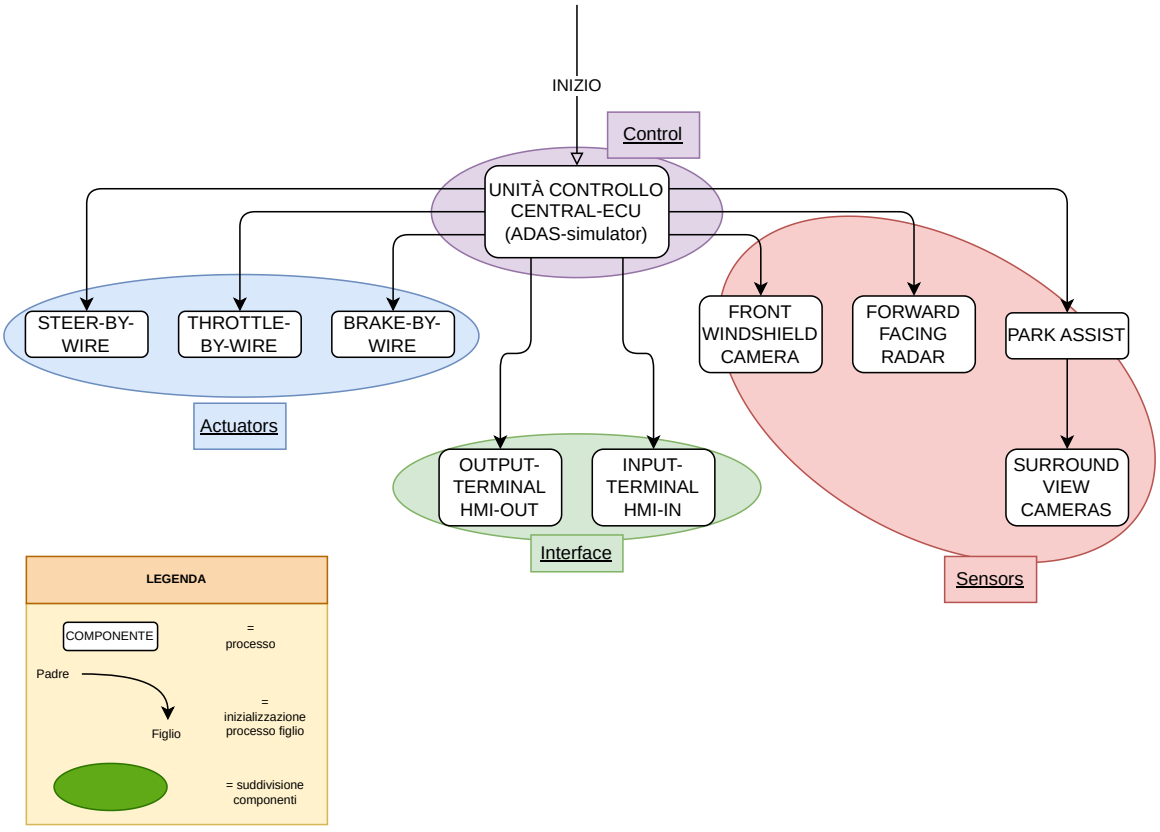

Figura 1: In figura è mostrata la gerarchia implementata nel programma, nel quale la *central-ECU* rappresenta il processo antenato di tutte gli altri processi.

#	Elemento facoltativo	Realizzato (SI/NO)	Descrizione dell'implementazione con indicazione del metodo/i principale/i
1	Ad ogni accelerazione, c'è una probabilità di 10^{-5} che l'acceleratore fallisca. In tal caso, il componente *throttle control* invia un segnale alla *central ECU* per evidenziare tale evento, e la Central ECU avvia la procedura di *ARRESTO*	SI	Metodo: throttle-control $\rightarrow$ throttle_failed(). Il metodo, tramite la funzione `rand()` della `libc` ottiene un numero aleatorio il cui modulo per 10000 simula una probabilita del 1 su 10^5 se eguagliato a 0[a]
2	Componente *forward facing radar*	SI	Sorgente *bytes-sensors.c*. Il processo esegue un ciclo infinito di letture, invii e scritture su file di log.
3	Quando si attiva l'interazione con park assist, la *Central ECU* sospende (o rimuove) tutti i sensori e attuatori, tranne *park assist* e *surround view cameras*	SI	Metodo: *central-ECU* $\rightarrow$ .... La metodo invia un segnale di terminazione immediata a tutti gli attuatori e ai sensori diversi da quelli specificati.
4	Il componente *park assist* non è generato all'avvio del sistema, ma creato dalla *Central ECU* al bisogno	SI	La *central-ECU* esegue la `fork` per la creazione di *park-assist* in ...
5	Se il componente *surround view cameras* è implementato, *park assist* trasmette a *Central ECU* anche i byte ricevuti da *surround view cameras*	SI	Nel ciclo principale di *park-assit* si esegue una `read` da *surround-view* e una `write` dei dati ricevuti sul pipe *assist.pipe*
6	Componente *surround view cameras*	SI	Sorgente *bytes-sensors.c*. Vedi facoltativo 2 - *forward facing radar*
7	Il comando di *PARCHEGGIO* potrebbe arrivare mentre i vari attuatori stanno eseguendo ulteriori comandi (accelerare o sterzare). I vari attuatori interrompono le loro azioni, per avviare le procedure di parcheggio	SI	Nella *central-ECU*, metodo ..., si esegue segnalazione di interruzione immediata che interrompe immediatamente i processi.
8	Se la *Central ECU* riceve il segnale di fallimento accelerazione da *throttle control*, imposta la velocità a 0 e invia all'output della *HMI* un messaggio di totale terminazione dell'esecuzione	SI	Nella *central-ECU*, una volta ricevuto il segnale di *PERICOLO* si esegue la procedura di arresto e si termina l'esecuzione ...

[a] La probabilita non risulta esattamente 10^-5 dato l'intervallo di valori producibili da `rand()`, ma la differenza dovrebbe essere trascurabile ai fini del progetto.

Descrizione architettura sistema

Nella seguente sezione viene presentata l'architettura del progetto e le scelte implementative prese, cercando di descrivere i motivi che hanno portato alle singole decisioni prese.

Implementazione componenti Segue una tabella nella quale si mostrano quali sorgenti implementano le componenti

Componente	Sorgente
Human-Machine Interface	hmi-input.c hmi-output.c
steer-by-wire	steer-by-wire.c
throttle control	throttle-control.c
brake-by-wire	brake-by-wire
front windshield camera	windshield-camera.c
forward facing radar	bytes-sensors.c
park assist	park-assist.c
surround view cameras	bytes.sensors.c
central ECU	central-ECU.c

Gerarchia del programma La gerarchia del progetto è stata ottenuta dalla richiesta cercando di massimizzare la semplicità.
Questa rappresenta lo scheletro del progetto e definisce quindi il flusso di lavoro dello stesso. In particolare si può notare in figura 1 che esistono due soli processi che inizializzano dei figli, ovvero l'unita di controllo centrale *central-ECU*, che rappresenta anche il programma genitore di tutto il sistema, e *park-assist*, che inizializza il suo unico figlio *surround-view cameras*.
È stato scelto di rendere l'unità di controllo genitore di tutto il sistema dato essere in collegamento con la quasi totalità dei processi agenti, cosicché il sistema fosse interamente inizializzato in un solo intervallo temporale da un unico processo.
Infatti nella *central-ECU* vengono immediatamente inizializzati i pipe, dopodiché vengono eseguite varie `fork` per la creazione e la connessione in lettura/scrittura ai pipe degli attuatori, delle due componenti relative alla *Human-Machine Interface* ed infine i due sensori *front windshield camera* e *forward facing radar*. L'unico componente figlio della *central-ECU* non immediatamente inizializzato rimane *park-assist*[6] il quale verrà messo in esecuzione dalla stessa unità di controllo quando questa riceverà un comando di *PARCHEGGIO* dall'interfaccia oppure dal sensore *windshield*.
Per limitare il tempo di inizializzazione della componente *park assist*, che a sua volta deve inizializzare il figlio *suround-view cameras* al proprio avvio e che quindi richiede qualche istante di tempo, e per poter mantenere semplice la struttura del sistema è stato deciso di rendere la *central-ECU* "server" nella connessione socket tra questa e *park assist*. In questo modo l'unità di controllo inizializzerà il server/socket `assist.sock` al momento dell'inizializzazione del sistema, non rendendo però necessario avviare la componente *park assist* fintantoché non ve ne sarà bisogno. Quando questa verrà avviata dovrà solo mettersi in connessione con la socket già creata precedentemente, risparmiando il tempo di

[6] La decisione di non inizializzare il componente immediatamente deriva dalla richiesta, ovvero dall'elemento facoltativo numero 4. Vedi la tabella di pagina 4.

inizializzazione della stessa.[7]
All'avvio della procedura di parcheggio, quindi, la *central-ECU* avvierà la componente *park-assist*, la quale si connetterà alla socket e inizializzerà *surround-view-cameras*. Le due componenti leggeranno per i 30 secondi successivi dati da sorgenti binarie e se in queste non risulteranno particolari pattern[8] allora l'esecuzione del parcheggio risulterà conclusa e il processo terminerà con successo, in caso invece almeno uno dei pattern succitati dovesse essere presente nei dati acquisiti dalle componenti allora la procedura di parcheggio terminerà immediatamente, ovvero non appena le uguaglianze vengono evidenziate, e si riavvierà immediatamente la procedura di parcheggio.

IPC nel sistema In figura 2, a pagina 7, si mostra una schematizzazione molto stilizzata della rete comunicativa del sistema. Il metodo di comunicazione maggiormente sfruttato all'interno del sistema è indubbiamente il **pipe** il quale struttura 8 canali di comunicazione su 9 (segnali esclusi). Il motivo della scelta dei **pipe** a discapito di altri metodi risiede nel fatto che le comunicazioni sono sostanzialmente unidirezionali (sensori $\rightarrow$ unità di controllo, unità di controllo $\rightarrow$ attuatori, con due evidenti eccezioni di *park assist* e di *surround-view-cameras*). La scelta ha permesso di implementare un sistema relativamente semplice, con un'unica **socket**, struttura più complessa da implementare e gestire.
Il protocollo di gestione dei canali risulta unico per tutte le tipologie di canali e per tutti i processi: il processo padre, l'unico processo che comunica con i propri processi figli, produce ed inizializza correttamente il file `.pipe` o `.sock` (rappresentante socket `UNIX`) nella directory *tmp* (dopo aver avuto l'accortezza di eliminare creazioni pendenti da vecchie esecuzioni tramite un `unlink`), il processo figlio si connette con l'adeguata procedura, differente tra pipe e socket, al canale di comunicazione nella fase inizializzazione del processo. Se la connessione non dovesse riuscire (se la `open` o la `connect` dovessero terminare con un errore) il figlio terminerebbe con codice di errore `EXIT_FAILURE` e stamperebbe lo stack dell'errore nello *stderr*.
Per l'implementazione del parcheggio il canale di comunicazione assume sembianze di una socket per evitare di dover terminare il programma *park-assist* ogni volta che la *central-ECU* riceve da questo uno dei pattern non ammissibili, che simulano una situazione di parcheggio non accettabile. Il protocollo prevede che ad ogni ciclo *park assist* riceva da *surround-view cameras* i byte da inviare alla *central-ECU* e che li invii, assieme da quelli letti dalla sorgente binaria. Se, eseguito l'invio, riceve dalla *central-ECU* un messaggio che indica la necessit di riavvio, allora ...

Gestione dei log Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

[7] Sebbene possa sembrare contro-intuitivo che la componente *park assist* rappresenti il client della connessione clinet/server (dato che sembrerebbe essere proprio *park assist* a svolgere un servizio per l'unità di controllo) risulta molto conveniente rendere la *central-ECU* il server poiché, essendo il canale comunicativo esclusivo per i due processi, non modifica il comportamento del sistema e rende l'inizializzazione del figlio più rapida.

[8] Per conoscere i pattern binari (espressi in codifica esadecimale) prendere visione del paragrafo 2, sotto-paragrafo *Componente central-ECU* della richiesta *Allegato_01.pdf*

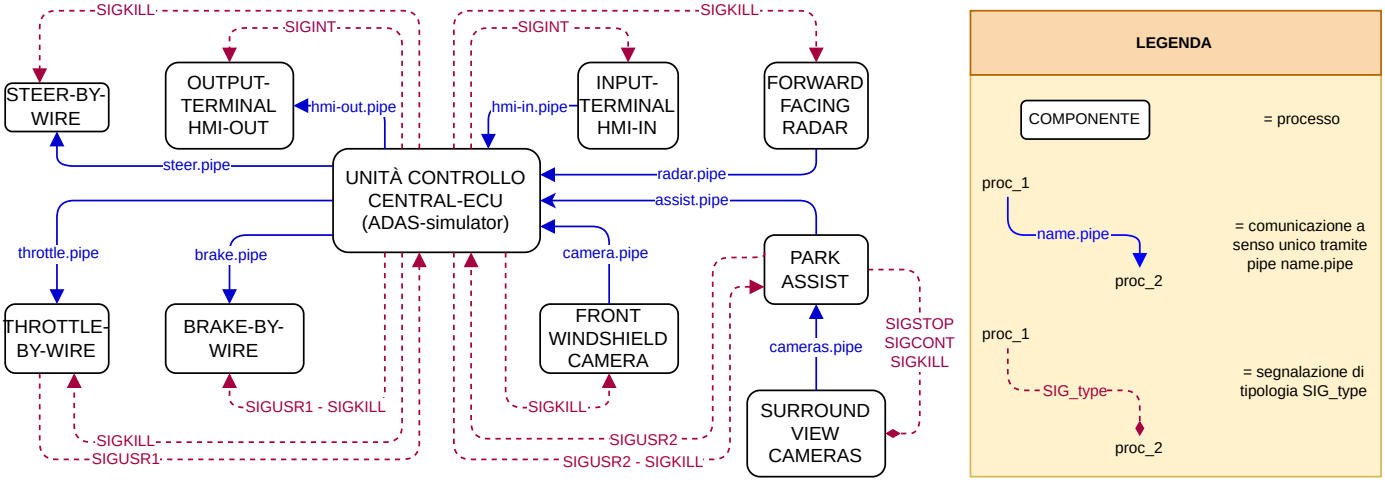

Figura 2: In figura sono rappresentati tutti e soli i percorsi comunicativi inseriti nel sistema: sono presenti 9 pipe e 15 percorsi di segnalazioni inviate dai processi del sistema ad altri processi dello stesso.

Nulla malesuada porttitor diam. Donec felis erat, congue non, volutpat at, tincidunt tristique, libero. Vivamus viverra fermentum felis. Donec nonummy pellentesque ante. Phasellus adipiscing semper elit. Proin fermentum massa ac quam. Sed diam turpis, molestie vitae, placerat a, molestie nec, leo. Maecenas lacinia. Nam ipsum ligula, eleifend at, accumsan nec, suscipit a, ipsum. Morbi blandit ligula feugiat magna. Nunc eleifend consequat lorem. Sed lacinia nulla vitae enim. Pellentesque tincidunt purus vel magna. Integer non enim. Praesent euismod nunc eu purus. Donec bibendum quam in tellus. Nullam cursus pulvinar lectus. Donec et mi. Nam vulputate metus eu enim. Vestibulum pellentesque felis eu massa.

Quisque ullamcorper placerat ipsum. Cras nibh. Morbi vel justo vitae lacus tincidunt ultrices. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. In hac habitasse platea dictumst. Integer tempus convallis augue. Etiam facilisis. Nunc elementum fermentum wisi. Aenean placerat. Ut imperdiet, enim sed gravida sollicitudin, felis odio placerat quam, ac pulvinar elit purus eget enim. Nunc vitae tortor. Proin tempus nibh sit amet nisl. Vivamus quis tortor vitae risus porta vehicula.

Fusce mauris. Vestibulum luctus nibh at lectus. Sed bibendum, nulla a faucibus semper, leo velit ultricies tellus, ac venenatis arcu wisi vel nisl. Vestibulum diam. Aliquam pellentesque, augue quis sagittis posuere, turpis lacus congue quam, in hendrerit risus eros eget felis. Maecenas eget erat in sapien mattis porttitor. Vestibulum porttitor. Nulla facilisi. Sed a turpis eu lacus commodo facilisis. Morbi fringilla, wisi in dignissim interdum, justo lectus sagittis dui, et vehicula libero dui cursus dui. Mauris tempor ligula sed lacus. Duis cursus enim ut augue. Cras ac magna. Cras nulla. Nulla egestas. Curabitur a leo. Quisque egestas wisi eget nunc. Nam feugiat lacus vel est. Curabitur consectetuer.

Suspendisse vel felis. Ut lorem lorem, interdum eu, tincidunt sit amet, laoreet vitae, arcu. Aenean faucibus pede eu ante. Praesent enim elit, rutrum at, molestie non, nonummy vel, nisl. Ut lectus eros, malesuada sit amet, fermentum eu, sodales cursus, magna. Donec eu purus. Quisque vehicula, urna sed ultricies auctor, pede lorem egestas dui, et convallis elit erat sed nulla. Donec luctus. Curabitur et nunc. Aliquam dolor odio, commodo pretium, ultricies non, pharetra in, velit. Integer arcu est, nonummy in, fermentum faucibus, egestas vel, odio.

Sed commodo posuere pede. Mauris ut est. Ut quis purus. Sed ac odio. Sed vehicula hendrerit sem. Duis non odio. Morbi ut dui. Sed accumsan risus eget odio. In hac habitasse platea dictumst. Pellentesque non elit. Fusce sed justo eu urna porta tincidunt. Mauris felis odio, sollicitudin sed, volutpat a, ornare ac, erat. Morbi quis dolor. Donec pellentesque, erat ac sagittis semper, nunc dui lobortis purus, quis congue purus metus ultricies tellus. Proin et quam. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos hymenaeos. Praesent sapien turpis, fermentum vel, eleifend faucibus, vehicula eu, lacus.

Indice